TRABAJO FIN DE GRADO

Optimización estructural mediante el uso acoplado de algoritmos genéticos y elementos finitos

Curso 2014-2015

Grado en Ingeniería Mecánica
Universidad Carlos III de Madrid

Departamento de Mecánica de Medios Continuos y Teoría de Estructuras

Realizado por:
Daniel Manzano Manzano

Tutor:
Alejandro Mahy Soler Trujillo
Ramón Zaera Polo

24 de Febrero de 2015

ÍNDICE

1 MOTIVACIÓN Y OBJETIVOS

1.1 MOTIVACIÓN

Las técnicas de optimización topológica en mecánica de estructuras ofrecen el ingeniero mecánico la posibilidad de determinar la geometría óptima sometida a restricciones de diseño y a unas condiciones de servicio determinadas. En particular, los algoritmos genéticos (AGs) presentan la ventaja, frente a las técnicas de optimización convencionales, de evitar el costoso cálculo del gradiente de la función. Es decir, los AGs tiene la ventaja de no necesitar un conocimiento específico de la función que se optimiza. Sin embargo, es muy importante el planteamiento de la función objetivo y de las restricciones, la identificación de las posibles soluciones y la codificación de éstas.

Los AGs son métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas. En ellos se mantiene una población que representa a un conjunto de posibles soluciones la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos y a un proceso de selección sesgado a favor de los mejores candidatos. Con los AGs se busca resolver diferentes problemas de optimización de manera que se obtengan soluciones iguales o mejores, pero de forma más rápida y sencilla que con otros métodos.

Actualmente los AGs se utilizan en diferentes campos y existe mucha bibliografía relacionada con su aplicación. Sin embargo, en la mayoría de los casos, la aplicación de los AGs no está descrita de forma detallada. Es decir, no se explica con claridad la codificación, ni la función objetivo, ni las restricciones utilizadas. Sólo se hace referencia al valor de los parámetros más importantes del algoritmo, como pueden ser, la probabilidad de cruce, mutación, etc. Esto, constituye un inconveniente para el usuario poco familiarizado con el tema.

Un dato importante es que el uso de los AGs requiere la utilización de un programa que permita ejecutar el algoritmo. Este programa puede ser escrito por el propio usuario o puede ser un programa comercial, como por ejemplo, la toolbox de MATLAB.

MATLAB es un programa matemático de ordenador con el cuál se pueden resolver problemas con diferentes herramientas. Una de estas herramientas o toolbox es la de algoritmos genéticos. Esta toolbox permite aplicar fácilmente los AGs a cualquier problema, visualizándose las iteraciones del algoritmo tanto de forma numérica como gráfica.

Por otro lado en el presente trabajo se hará uso del método de elementos finitos para análisis estructural. El método de elementos finitos (MEF) es un método numérico general para la aproximación de soluciones de ecuaciones diferenciales parciales muy utilizado en diversos problemas de ingeniería. El MEF está pensado para ser usado en computadoras y permite resolver ecuaciones diferenciales asociadas a un problema físico sobre geometrías complicadas, al igual que con los algoritmos genéticos requiere la utilización de un programa que permita ejecutarlo, como por ejemplo, el código de elementos finitos ABAQUS,

ABAQUS constituye una herramienta eficaz y muy customizable para crear modelos de elementos finitos de forma interactiva, visualizar resultados de los análisis y la automatizar de procesos mediante scripts o subrutinas en lenguaje Python.

Los usuarios de ABAQUS puede crear el modelo de elementos finitos a partir de geometría creada en el propio programa o importarla directamente a partir de los formatos CAD más comunes, o también importar directamente la malla de otro software de elementos finitos.

1.2 DESCRIPCIÓN DEL PROYECTO Y OJBETIVOS

El presente proyecto tiene como objeto desarrollar una herramienta que permita diseñar una estructura sencilla mediante el uso de la interfaz gráfica de usuario de MATLAB, donde se puedan elegir entre distintos tipos apoyos, cargas y perfiles. Una vez realizado el diseño deseado la herramienta debe de ser capaz de optimizar dicho perfil de manera que nos quede una estructura con el menor peso posible cumpliendo una serie de condiciones restrictivas. Esta optimización se llevará a cabo mediante el uso acoplado del toolbox de Matlab para algoritmos genéticos, y el código de elementos finitos ABAQUS. El proceso de optimización se realizará atendiendo al peso mínimo de la estructura y se deberán cumplir dos restricciones ligadas a evitar la plastificación de la viga y la superación de un desplazamiento máximo.

El tipo de estructuras a diseñar serán vigas continuas lineales y las cargas se aplicaran en el plano y/o pares perpendiculares a él.

Para ello el problema en ABAQUS quedará definido mediante un Script en lenguaje Python de manera que se pueda modificar y leer las variables del problema mediante MATLAB interactuando ambos programas para la optimización de la estructura. Se utilizará una hoja de Excel como enlace de intercambio de información entre el script de ABAQUS, el programa en MATLAB y la interfaz gráfica de usuario.

A continuación se hará una introducción del funcionamiento de los AGs y del método de elementos finitos así como de los programas de cálculo que se utilizaran.

2 ANTECEDENTES

2.1 ALGORITMOS GENÉTICOS

2.1.1 INTRODUCCIÓN A LOS ALGORITMOS GENÉTICOS

Los Algoritmos Genéticos (AGs) *"son métodos estocásticos de búsqueda ciega de soluciones cuasi-óptimas",* ya que asignan probabilidades de selección o puntuaciones a los elementos de una población base en función de su aptitud y no se dispone de ningún conocimiento específico del problema sino que la búsqueda se basa exclusivamente en los valores de la función objetivo con lo que se consigue una gran amplitud de búsqueda. *"En ellos se mantiene dicha población que representa a un conjunto de posibles soluciones la cual es sometida a ciertas transformaciones con las que se trata de obtener nuevos candidatos y a un proceso selección sesgado a favor de los mejores candidatos*" (C.Cervigón, L. Araujo: 2010). Al trabajan con una población se reduce la posibilidad de alcanzar un falso óptimo.

Los AGs procesan externamente cadenas de códigos, es decir, es un método de búsqueda codificada ya que trabaja con cadenas de códigos siendo más usada la binaria (0 y 1). Sin embargo, lo que se está produciendo internamente son similitudes entre cadenas de forma que al tratar cada una de las cadenas de la población se están procesando a la vez todos los patrones de similitud que contienen, que son muchos más (Teorema fundamental de los Algoritmos genéticos conocido como Teorema de Holland, Pérez, A. (1999)). Es por esta propiedad por la que los AGs son mucha más eficaces que otros métodos de búsqueda ciega

2.1.2 BREVE RESEÑA HISTÓRICA

Los AGs fueron inventados en 1975 por John Holland, de la Universidad de Michigan. Los mecanismos de los que se valen los AG para llevar a cabo esta búsqueda pueden verse como una metáfora de los procesos de evolución biológica propuestos por Charles Darwin en su obra, "El origen de las especies".

John Holland fue el primero que intentó realizar pequeños modelos de la naturaleza que tuvieran algunas de sus peculiaridades y ver cómo funcionaban pero fue a principios de los 60, en la Universidad de Michigan en Ann Arbor, donde sus ideas comenzaron a desarrollarse y a dar frutos. Aprendió que la evolución era una forma de adaptación más potente que el simple aprendizaje, y tomó la decisión de aplicar estas ideas para desarrollar programas bien adaptados para un fin determinado. En esta universidad, Holland impartía un curso titulado teoría de sistemas adaptativos. Dentro de este curso, y con una participación activa por parte de sus estudiantes, fue donde se crearon las ideas que más tarde se convertirían en los AG.

Más adelante Holland, con la ayuda de sus alumnos, en especial de David Goldberg, consiguieron escribir un AG en un ordenador personal y junto a otras aplicaciones convirtieron a los AG en un campo con bases suficientemente aceptables como para celebrar la primera conferencia en 1985, ICGA´85.

2.1.3 ESTRUCTURA DE LOS ALGORITMOS GENÉTICOS

La estructura de los AGs se fundamenta en las teorías evolutivas .Se parte de una población inicial normalmente aleatoria (aunque ésta incluso siendo aleatoria puede estar restringida a ciertos valores como ocurrirá con nuestros problemas) de donde se seleccionarán a un grupo de individuos que formarán el grupo de los progenitores o población intermedia, este proceso se conoce como proceso de selecci'on. Sobre esta población de progenitores se llevan a cabo operadores genéticos de cruce y mutación para finalmente realizar el remplazo de la población inicial eligiendo individuos de la población intermedia y de la población de descendientes. En nuestro problema cada individuo hará referencia a una estructura distinta.

Este bucle de selección-cruce-mutación-reemplazo se repite normalmente un número determinado de generaciones, hasta alcanzar la solución óptima Ver figura 2.0

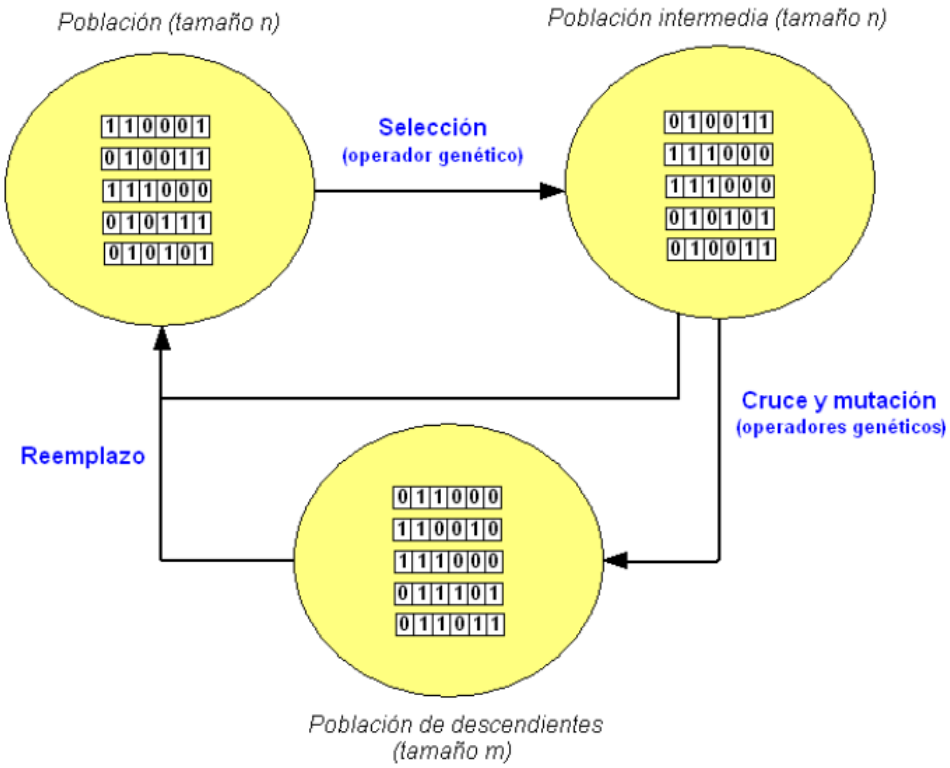


Figura 2.1.3.1: Bucle selección-cruce-mutación-reemplazo.

2.1.4 CODIFICACIÓN

En los algoritmos genéticos cuando hablamos de un individuo estamos hablando de una cadena de bits, normalmente en numeración binaria. Cada variable del problema a resolver que normalmente será un número entero o decimal se identifica con un gen. El conjunto de genes (parámetros de todas las secciones posibles) da lugar a un cromosoma. Para definir un problema debemos definir antes la longitud que tendrá la cadena de bits de cada individuo. Cada solución por tanto debe tener una única representación binaria.

La figura 2.1 muestra un ejemplo de codificación para un problema de dos variables x1 y x2 .

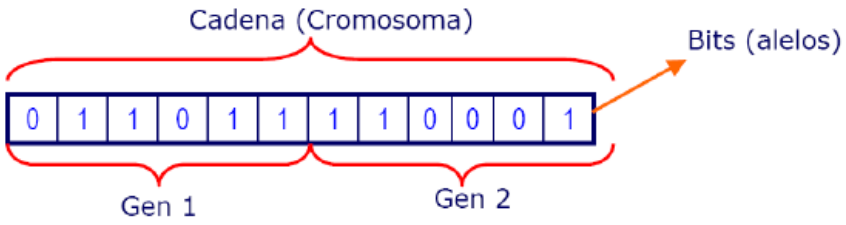


Figura 2.1.4.1: Codificación binaria de dos variables

.

Dependiendo de si el tipo de variables con las que trabajamos en nuestro problema son enteras o decimales éstas podrán tomar un mayor o menor número de valores. En este caso las variables han sido representadas mediante 6 bits, por tanto, en caso de corresponder con una variable entera cada una puede tomar 2^6 valores distintos. En el algoritmo genético que se utiliza en esta herramienta se trabajará con variables decimales.

Cuando el dominio del problema son números decimales, se debe discretizar el intervalo de soluciones posibles y hay que tener en cuenta la precisión, de modo que la distancia entre puntos consecutivos del intervalo [x_{min}, x_{max}] sea menor que la tolerancia especificada.

Una forma habitual de realizar esta discretización se explica a continuación. Por ejemplo, para soluciones comprendidas en el intervalo [0,1] con precisión m=0.005, la longitud L de la representación binaria será calculada como:

$$\frac{x_{max} - x_{min}}{2^L} < m \rightarrow L = log_2\left(1 + \frac{x_{max} - x_{min}}{m}\right)$$

Ecuación 2.1.4.1

En este caso L=8, siendo 0 = x_{min} y 1 = x_{max}. La precisión es exactamente de $1(2^8\text{-}1) = 0.0039$; está por debajo de la precisión dada por lo que es válida y es posible representar $2^8 = 256$ soluciones.

La forma con la cual se decodifica los números decimales es la siguiente. Se tiene por ejemplo:

1 0 1 1 1 0 1 0 = v = 184 pero representa a x = 0.7294. Para sacar x usamos la siguiente expresión.

$$x = x_{min} + bin2dec(v)\left(\frac{x_{max} - x_{min}}{2^L - 1}\right)$$

Ecuación 2.1.4.2

2.1.5 CREACIÓN DE POBLACIÓN INICIAL

La creación de la población inicial se genera normalmente de manera aleatoria aunque si se tiene información de las posibles soluciones es conveniente crear una población inicial con individuos que pudieran ser una solución factible. De no ser así podría ser conveniente crear una población inicial con mayor número de individuos para favorecer la diversidad. En nuestra herramienta crearemos poblaciones de 30 individuos debido a que tendremos muchos genes pero los valores suelen estar comprendidos entre 50 y 100 individuos.

2.1.6 FUNCIÓN DE EVALUACIÓN Y DE APTITUD

La función de aptitud (fitness), no es más que la función objetivo de nuestro problema de optimización. Una característica que debe tener esta función de aptitud es la de castigar las malas soluciones y premiar las buenas, para valorar qué tan buenos son los individuos. En nuestro problema la función objetivo será la masa de la estructura y la función de aptitud será la misma pero sumando penalizaciones. Éstas se originan como consecuencia de que un punto o varios puntos de la viga superan el límite elástico del material o el desplazamiento máximo establecido.

2.1.7 SELECCIÓN

El proceso de selección consiste en escoger a individuos como progenitores y se basa en la aptitud de los individuos, es decir, los individuos que sean mejores tendrán una mayor probabilidad de ser seleccionados para la reproducción. Este primer proceso no supone la aparición de nuevos individuos sino que forma una población intermedia como ya se vio en el apartado de ESTRUCTURA DEL ALGORITMO.

Los métodos de selección más utilizados suelen ser estocásticos. A continuación se definen algunos métodos de selección:

2.1.7.1 Procedimientos de selección.

Métodos aleatorios simples o equiprobable: El proceso de selección se produce sin tener en cuenta los mejores individuos, se asigna la misma probabilidad de selección a todos los individuos de la población inicial.

Métodos estocásticos:

- Método por sorteo.

- Método universal o por ruleta.

- Método jerárquico.

- Método por torneos.

En nuestra herramienta se ha utilizado un método estocástico uniforme para la resolución de los problemas que se han planteado.

2.1.8 CRUCE

La operación de cruce (junto con la de mutación) es posiblemente la más importante debido a que realiza la combinación entre dos individuos para crear dos nuevos descendientes sin que desaparezca la pareja de progenitores. Esta operación consigue obtener a los mejores individuos de la población. El cruce se realiza con un intercambio de bits entre ambos progenitores. Se establece la probabilidad de cruce "cp" y se le da aleatoriamente a los individuos valores comprendidos entre 0 y 1, "b", de manera que si b>cp se cruzan.

Existen distintos tipos de cruce, dependiendo del número de puntos en que se corta la cadena de bits, en nuestra herramienta se ha utilizado el cruce disperso (`crossoverscattered`) que consiste en que se realizan varios cortes en la cadena de forma que el problema no se homogeneizaba tan pronto como ocurría con el cruce monopunto.

2.1.9 MUTACIÓN

En teoría la mutación no es un operador genético tan importante como el cruce ya que la probabilidad de que se aplique a un individuo es mucho menor que la del cruce pero sin embargo se ha comprobado que un AG puede funcionar sin cruces, pero no sin mutación ya que se originan pérdidas de alelos que no se podrían recuperar (incesto) y esto se evita con la mutación. Es más, en nuestra herramienta utilizamos un alto grado de mutación para evitar la rápida convergencia de nuestro problema.

Los operadores de mutación abren el abanico a posibles soluciones fuera del rango inicial, es por eso que se denominan operadores de exploración. La mutación se fundamenta en modificar algunos de los bits de un individuo o incluso el total. Si aumentamos mucho la probabilidad de mutación y disminuimos la probabilidad de cruce haremos demasiado aleatoria la búsqueda mientras que de la manera opuesta explotaremos las mejores soluciones pero debemos tener en cuenta que una mutación muy baja puede llevarnos a mínimos locales.

Para realizar la mutación se define, para cada bit, una probabilidad de mutación: p_m. Luego, se asigna aleatoriamente un número "c" entre [0,1] a cada gen y si c< p_m, entonces se aplica mutación, es decir el bit cambia de valor. En la figura 2.1.9.1 se muestra un ejemplo.

Figura 2.1.9.1: Mutación

2.1.10 REEMPLAZO

El reemplazo es la operación final correspondiente a una iteración, es el encargado de seleccionar los individuos que formarán parte de la siguiente generación. La nueva generación debe tener en mismo número de individuos que la población inicial (n individuos). Por un lado tenemos una población intermedia del proceso de selección (N individuos) y por el otro los resultantes de la operación de

cruce y mutación (m individuos). La operación de reemplazo deberá seleccionar de entre estos los más aptos. Hay diferentes formas de realizar el reemplazo y esta afirmación anterior no siempre se cumple:

Reemplazo al vuelo o inmediato: Los m descendientes sustituyen directamente a sus m progenitores sin ninguna consideración previa.

Reemplazo por factor de llenado: Los m descendientes sustituyen a aquellos individuos de la población intermedia a los que más se asemejan.

Reemplazo por inserción: "Este reemplazo se puede realizar de dos maneras:

- El número de descendientes es mayor que la población inicial, por lo que la operación de reemplazo se realizará únicamente con descendientes de forma que ningún individuo podrá prevalecer más de una generación

- El número de descendientes es menor que el tamaño de la población inicial por lo tanto todos los descendientes pasarán a formar parte de la nueva generación junto con los mejores de la población intermedia.

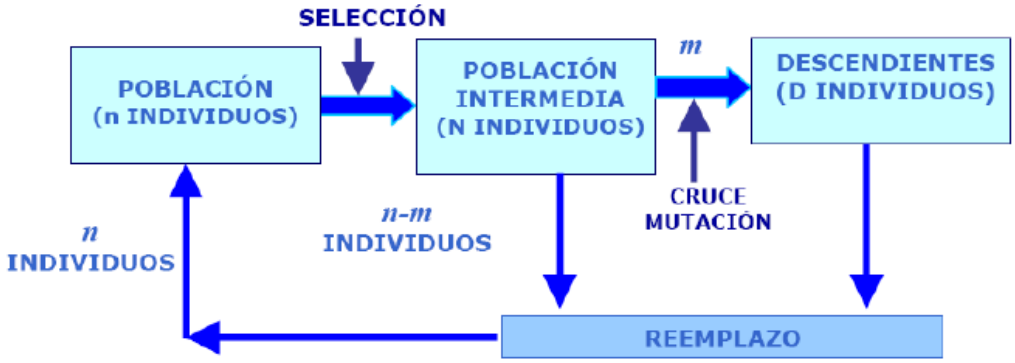


Figura 2.1.10.1: Reemplazo

2.1.11 CRITERIO DE PARADA

El criterio de parada más utilizado suele ser el número máximo de iteraciones de algoritmo, pero no siempre nos garantiza buenos resultados porque es posible que el algoritmo no haya llegado a converger o no haber alcanzado el óptimo. Por esta razón se suele utilizar un criterio de homogeneidad de la población o el que utilizamos en nuestra herramienta de optimización: Se establece un valor "T" o de tolerancia, de manera que si en un número determinado de iteraciones (también previamente establecido) no mejora el valor "T" el algoritmo se detiene.

2.2 ALGORITMOS GENÉTICOS CON MATLAB

Existen varias posibilidades de utilizar algoritmos genéticos utilizando MATLAB, entre ellas están: crear tu propio código, utilizar la toolbox de optimización mediante ags o por último y el que hemos elegido para realizar nuestra herramienta, utilizar los comandos y funciones intrínsecas en el programa, que es lo mismo que utilizar la toolbox pero de esta forma conseguimos que nos quede una herramienta automatizada sin necesidad de establecer los parámetros y operadores genéticos cada vez que queramos resolver un problema. Cómo se utilizan los AGs con MATLAB viene explicado en el apartado 4.3.

2.3 ELEMENTOS FINITOS

Los elementos finitos son un método numérico para aproximar soluciones de ecuaciones o sistemas de ecuaciones diferenciales recurriendo a un proceso de discretización del dominio.

El objetivo es transformar un sistema continuo regido por una ecuación o sistema de ecuaciones diferenciales que tendrá infinitos grados de libertad a un sistema donde los grados de libertad sean finitos, de ahí el nombre del método. En la figura siguiente podemos ver como se realiza el proceso.

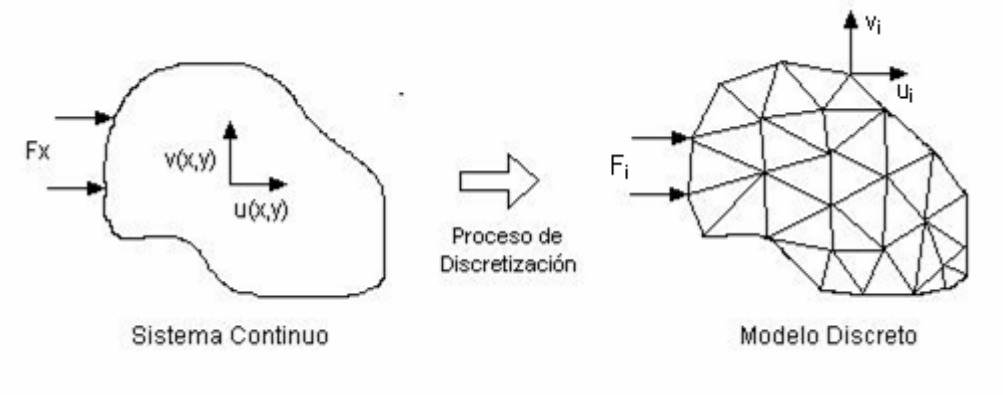


Figura 2.3.1: Proceso de Discretización.

El sistema de elementos estará regido por un sistema de ecuaciones. En un sistema se pueden distinguir las siguientes partes:

- **Dominio**. Es el lugar geométrico del espacio donde se va a resolver el sistema. El dominio de un sistema se divide en diferentes tipos de elementos dependiendo del tipo de problema, por ejemplo si el problema es lineal se divide mediante puntos (como ocurrirá en nuestro caso).
- **Condiciones de contorno**. Son las variables que conocemos y que pueden alterar el sistema: cargas, desplazamientos, temperaturas, voltaje, focos de calor,...
- **Incógnitas**: Son las variables que se desconocen y que deseamos calcular como desplazamientos, temperaturas…

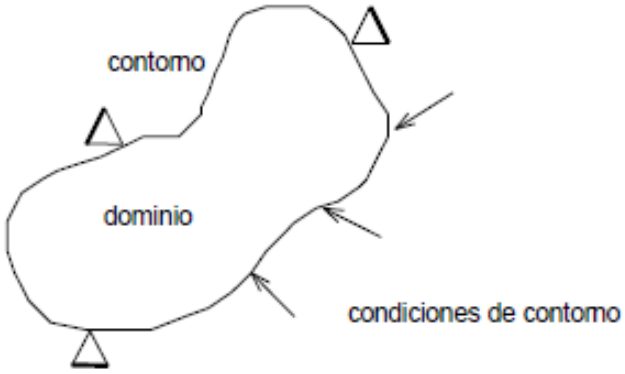


Figura 2.3.2: Contorno, dominio y condiciones de contorno

Cada elemento en que se divide el sistema está representado por una serie de puntos llamados nodos que conectan entre sí los elementos. Sobre estos nodos se materializan las incógnitas fundamentales del problema.

A la hora de analizar elementos estructurales las incógnitas que tendremos serán los grados de libertad de los nodos, es decir, serán los desplazamientos de cada nodo y a partir de ellos podremos calcular el resto de incógnitas tales como tensiones deformaciones…

Por ejemplo nosotros analizaremos una viga en voladizo con una carga puntual en el extremo como la que podemos ver en la figura 2.3.3

Figura 2.3.3: Viga en voladizo

Y la discretización será de la siguiente manera: Para nuestros problemas se programará que una viga puede tener diferentes secciones de un mismo perfil. La viga se discretizará en cuatro nodos por cada una de sus secciones. (Puede verse la programación en el apartado Módulo MESH correspondiente al Script de ABAQUS). (Figura 2.3.3):

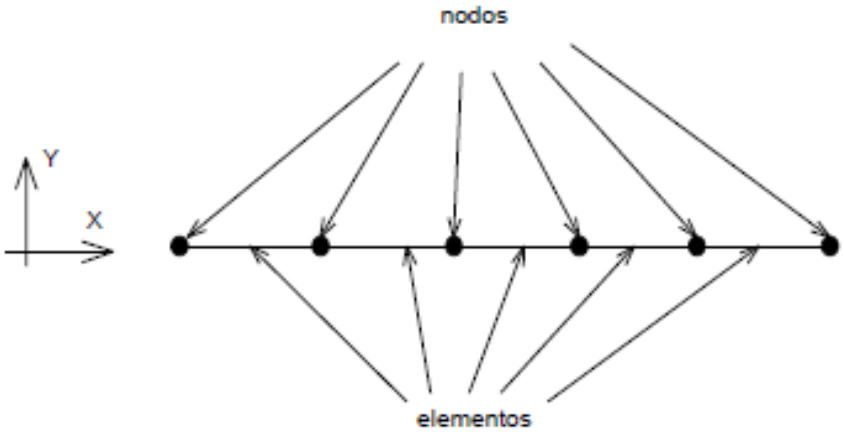


Figura 2.3.3: Viga con dominio discretizado

Los grados de libertad de cada nodo serán:

- Desplazamiento en dirección x
- Desplazamiento en dirección y
- Giro según z

A las fórmulas que relacionan el comportamiento en el interior del continuo con el valor que adoptan los grados de libertad se llega mediante las funciones de interpolación, "que interpolan el valor de la variable nodal dentro del elemento".

2.3.1 PRINCIPIOS GENERALES APLICADOS A UN CONTINUO ELÁSTICO

A continuación se muestran algunas de las ideas básicas relacionadas con los fundamentos matemáticos del MEF aplicadas al caso estructural.

2.3.1.1 Ecuaciones de equilibrio. Principio de los trabajos virtuales

En muchas ocasiones nos encontramos con problemas regidos por ecuaciones diferenciales y condiciones de contorno sobre las incógnitas muy difíciles de resolver, la aplicación del MEF nos permite encontrar una solución aproximada apoyándose en la expresión integral del Principio de los Trabajos Virtuales.

"Se considera un continuo elástico como el de la figura sometido a unas fuerzas superficiales $\{t\}=\{t_x\ t_y\ t_z\}^T$ y a unas fuerzas por unidad de volumen $\{X\} = \{X_x\ ,\ X_y\ ,\ X_z\}^T$, (las fuerzas por unidad de superficie podrían ser presiones y el peso propio sería una fuerza por unidad de volumen). El vector desplazamientos lo notamos por $\{U\} = \{u,v,w\}^T$. Las deformaciones correspondientes a estos desplazamientos:

$$\{\varepsilon\} = \{\varepsilon_{xx}, \varepsilon_{yy}, \varepsilon_{zz}, \gamma_{xy}, \gamma_{yz}, \gamma_{zx}\}^T$$

T y las tensiones debidas a estas deformaciones serán:

$$\{\sigma\} = \{\sigma_{xx}, \sigma_{yy}, \sigma_{zz}, \tau_{xy}, \tau_{yz}, \tau_{zx}\}^T$$

Las ecuaciones de equilibrio para un elemento diferencial de volumen pueden escribirse de la forma:

$$\nabla\{\sigma\} + \{X\} = 0$$

"donde el vector {X} incluye de forma general las fuerzas de inercia {X} = {X} – ρü, es decir, consideramos las fuerzas de inercia como fuerzas por unidad de volumen. Multiplicando esta ecuación por una función de ponderación {du} e integrando:"

$$\int_v \{\delta u\}^T \nabla\{\sigma\} dv + \int_v \{\delta u\}^T \{X\} dv = 0$$

Ecuación 2.3.1

"Utilizando la formula de Green se puede escribir:"

$$-\int_v \nabla\{\delta u\}^T \{\sigma\} dv + \int_s \{\delta u\}^T \{\sigma\} \bar{n} ds + \int_v \{\delta u\}^T \{X\} dv = 0$$

Ecuación 2.3.2

"Si se asocia la función de ponderación {δu} con un desplazamiento virtual, el operador ∇ actuando sobre él será una deformación virtual:"

$$\{\delta e\} = \nabla\{\delta u\}$$

"El equilibrio en el contorno exige que se cumpla la relación $\{\sigma\} \cdot \{n\} = \{t\}$ y sustituyendo en la expresión 2.3.2:"

$$-\int_v \nabla\{\delta\varepsilon\}^T \{\sigma\} dv + \int_s \{\delta u\}^T \{t\} ds + \int_v \{\delta u\}^T \{X\} dv = 0$$

Ecuación 2.3.3

"En la relación anterior es posible introducir la ley de comportamiento de material mediante la matriz elástica con las propiedades de éste, [C], de forma que:"

$$\{\sigma\} = [C](\{\varepsilon\} - \{\varepsilon_0\}) + \{\sigma_0\}$$

Ecuación 2.3.4

"siendo los vectores $\{\varepsilon_0\}^T$ y $\{\sigma_0\}^T$ las deformaciones y las tensiones iniciales respectivamente. Introduciendo la expresión 2.6), suponiendo deformaciones y tensiones iniciales nulas, en la ecuación (2.5), obtenemos:"

$$\int_v \{\delta\varepsilon\}^T [C]\{\varepsilon\} dv = \int_v \{\delta u\}^T (\{X\} - \rho\{\ddot{u}\}) dv + \int_s \{\delta u\}^T \{t\} ds$$

Ecuación 2.3.5

"que constituye la formulación del PTV y relaciona el sistema de cargas real y esfuerzos con el virtual de desplazamientos."

En nuestro proyecto no se consideran las fuerzas de inercia.

2.3.1.2 Funciones de interpolación

"Una vez dividido el sistema en nodos se toma un conjunto de funciones que definen de manera única el campo de desplazamientos dentro del elemento en función de los desplazamientos en los nodos del mismo:"

$$\{u(x,y,z)\} = [N(x,y,z)]\{U\}$$

"Siendo {U} el vector con los desplazamientos nodales. Una vez conocidos los desplazamientos en todos los nodos se determinan las deformaciones "

$$\{\varepsilon\} = [D]\{u\}$$

"donde [D] es el operador diferencial que depende del problema en estudio."

"Sustituyendo el valor del desplazamiento tenemos que:"

$$\{\varepsilon\} = [D][N]\{U\} = [B]\{U\}$$

"donde se obtiene el valor de las deformaciones en función de los desplazamientos nodales.Sustituyendo la ecuación anterior en la expresión del PTV (2.7 tenemos:"

$$-\int_v [B]^T [C][B]\{U\} dv + \int_v [N]^T (\{X\} - \rho[N]\{\ddot{U}\}) dv + \int_s [N]^T \{t\} ds$$

Ecuación 2.3.6

"Reordenando esta ecuación podemos llegar a un sistema de la forma"

$$[M]\{\ddot{U}\}+[K]\{U\} = \{P\}$$

"donde se definen:"

- **Matriz de masa consistente**

$$[M] = \int_v [N\,]^T \rho [N] dv$$

- **Matriz de rigidez**

$$[K] = \int_v [B]^T [C][N] dv$$

- **Matriz de cargas nodales consistentes**

$$\{P\} = \int_v [N]^T \{X\} dv + \int_s [N]^T \{t\} ds$$

"La expresión anterior es general y permite determinar las matrices elementales para cualquier tipo de discretización. Estas matrices se deben obtener para todos los elementos."

FUENTE: A. Carnicero. *Introducción al método de elementos finitos.*

2.3.1.3 Ajustes de las características globales y solución del sistema.

Todas las matrices deben estar referenciadas al mismo sistema de coordenadas. Cada elemento puede poseer su propio sistema de referencia unitario y debemos transformarlos al sistema de referencia global. Este procedimiento se realiza antes del ensamblaje de todas ellas. A continuación se suma en cada posición nodal el aporte realizado por los distintos elementos y se aplican las restricciones de los desplazamientos nodales que conocemos. Resolviendo este sistema tendremos la solución.

2.4 BREVE INTRODUCCIÓN A ABAQUS

ABAQUS será el programa que utilizaremos para revelar las incógnitas de tensiones y desplazamientos que se producirán en nuestras estructuras. Es un programa destinado a resolver problemas de ingeniería por el MEF, tiene potencial tanto para resolver problemas lineales como no lineales.

Como características a destacar podemos decir que ABAQUS no utiliza sistema de unidades por los que es conveniente asignar un criterio antes de comenzar a trabajar con él. En nuestra herramienta se insertarán todas las unidades en el SI.

Para trabajar con ABAQUS utilizaremos un archivo de entrada que será un Script programado en lenguaje Python donde se desarrollarán todos los módulos del programa necesarios para la elaboración de la Estructura. A continuación se muestra una figura de cómo se realizará el Análisis.

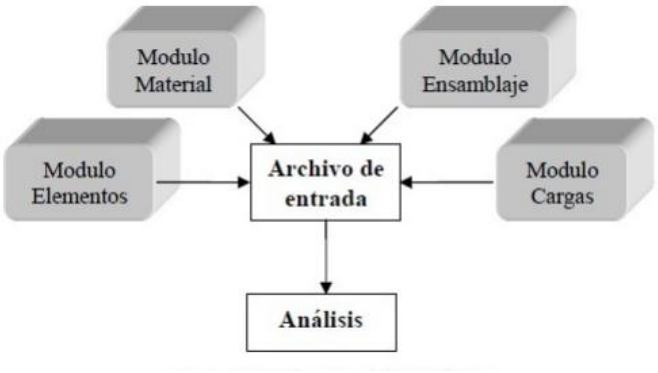


Figura 2.4.1: Componentes de un modelo en ABAQUS

El propósito que se busca al utilizar ABAQUS es predecir la respuesta de un modelo sometido a diferentes condiciones y cargas. El análisis en este programa de elementos finitos está basado en el concepto de STEPS. A través de estos se introducen los cambios en las condiciones de contorno, en las cargas o en el tipo de análisis.

En los siguientes apartados se hablará más del funcionamiento de ABAQUS, ya que no se utilizará el programa como tal para diseñar las estructuras, sino que se hará a través de un Script en lenguaje Python compatible con ABAQUS y se irán explicando paso a paso los comando aplicados y cómo sería en caso de utilizar directamente el programa.

3 GENERACIÓN AUTOMATIZADA DEL MODELO

La generación automatizada del modelo y correcto funcionamiento de la herramienta estará basada en la buena comunicación entre los distintos programas que se van a utilizar. En la Figura 3.6.1 se muestra un esquema de cómo estarán conectados los distintos programas.

3.1 PROGRAMAS EMPLEADOS PARA LA EJECUCIÓN Y DESARROLLO DE LA HERRAMIENTA.

En primer lugar se instalarán todos los programas necesarios en el PC. Se instalarán los programas en el siguiente orden para evitar problemas futuros:

1. Microsoft Office (Excel).
2. ABAQUS 6.9-2SE (Versión Estudiante)
3. MATLAB
4. Python 2.7.8
5. Módulos de Python

Una vez instalados todos los programas se procede a la generación de los distintos Scripts cuya construcción y funcionamiento se explicará detalladamente en los siguientes apartados.

3.2 PARÁMETROS A DECIDIR POR EL USUARIO O PARÁMETROS DE ENTRADA

El usuario podrá elegir y editar los siguientes parámetros para diseñar la estructura. El número de elementos se calculará automáticamente de forma indirecta. Se hará más hincapié en cómo se introducen los parámetros en el apartado de Interfáz Gráfica de Usuario. Estos son los parámetros a introducir en la interfaz:

PARÁMETROS GENERALES:

- Longitud
- Nº Discretización
- Densidad
- Módulo de Young
- Coeficiente de Poisson
- Tipo de Perfil
- Límite Elástico

NÚMERO DE ELEMENTOS (CARGAS Y COACCIONES)

- Nº Cargas Puntuales
- Nº Cargas Lineales
- Nº Momentos Puntuales
- Nº Apoyos Simples
- Nº Empotramientos

CARGAS:

- Valores Carga Puntual
- Posiciones Carga Puntual
- Valores Carga Lineal
- Posiciones Inicial Carga Lineal
- Posiciones Final Carga Lineal
- Valores Momento Puntual
- Posiciones Momento Puntual

COACIONES:

- Posiciones Apoyo Simple
- Posiciones Empotramientos

3.3 RESULTADOS: PARÁMETROS A OPTIMIZAR, O PARÁMETROS DE SALIDA

Como ya hemos mencionado el objetivo de la herramienta es conseguir que con los parámetros de entrada introducidos obtener una estructura en la que se emplee la menor cantidad de material para construirla o dicho de otra forma, que su masa sea la mínima posible. Por lo tanto el parámetro a optimizar será el peso de la estructura, llamado Función Objetivo según la nomenclatura de los Algoritmos Genéticos. Puesto que hay diferentes tipos de perfiles y habrá más de un tramo distinto (pero con el mismo perfil) para cada estructura el resultado del problema serán los parámetros de cada uno de ellos así como el peso final de la viga.

El formato en el que se obtendrán los resultados será por un lado mediante muestra en pantalla en la ventana de comando de MATLAB donde quedará en último lugar el peso y por otro lado se guardarán los valores en archivos con extensión .txt.

El script AGsProyecto.m tiene establecida la opción de mostrar al mejor individuo y la media de cada iteración para hacer un seguimiento a tiempo real del funcionamiento del algoritmo y tener así una gráfica del comportamiento de su evolución cuando este termine.

3.4 SCRIPTS Y ARCHIVOS CREADOS.

Los Scripts acompañados de los programas con los que se crean son los Siguientes:

1. InterfazGraficaProyecto.m → Script creado en MATLAB que va acompañado de otro archivo, InterfazGraficaProyecto.fig.
2. ExcelProyecto.xls → Hoja Excel creada con Microsoft Excel. Es el archivo donde se almacenan los datos que elige el usuario y de los que dependen los demás Scripts.
3. AGsProyecto.m → Script creado con MATLAB que corresponde al Script de Algoritmos Genéticos.
4. ScriptProyecto.m → Script creado con MATLAB que corresponde al Script de Función Objetivo y aplicación de Restricciones.
5. ScrpitFinal.py → Script creado con Microsoft Visual Studio 2008 escrito en lenguaje Python.

3.5 ARCHIVOS QUE SE CREAN DURANTE EL FUNCIONAMIENTO

Durante la ejecución de la herramienta se crean y eliminan una serie de archivos temporales. Por un lado se encuentran los archivos que almacenan los valores de los parámetros de las secciones, es decir, aquellos que se van a optimizar por lo que serán distintos para cada individuo e irán cambiando en cada iteración. Estos archivos son creados por AGsProyecto y son los siguientes.

1. Sección Tubular.
   - resultadosrs.txt → Archivo con los valores del radio exterior de cada tramo de viga en caso de seleccionar sección tubular.
   - resultadosts.txt → Archivo con los valores del espesor de cada tramo de viga en caso de seleccionar sección tubular.

2. Sección Rectangular.

   - resultadosar.txt → Archivo con los valores de la altura de cada tramo de viga en caso de seleccionar sección rectangular.

   - resultadosbr.txt → Archivo con los valores de la base de cada tramo de viga en caso de seleccionar sección rectangular.

3. Sección Caja. .

   - resultadosac.txt → Archivo con los valores de la altura de cada tramo de viga en caso de seleccionar sección caja.

   - resultadosbc.txt → Archivo con los valores de la base de cada tramo de viga en caso de seleccionar sección caja.

   - resultadostc.txt → Archivo con los valores del espesor de cada tramo de viga en caso de seleccionar sección caja.

4. Sección Doble T.

   - resultadosh0.txt → Archivo con los valores de la altura del alma cada tramo de viga en caso de seleccionar sección Doble T.

   - resultadosh1.txt → Archivo con los valores de la altura del canto de cada tramo de viga en caso de seleccionar sección Doble T.

   - resultadosb0.txt → Archivo con los valores del ancho del alma de cada tramo de viga en caso de seleccionar sección Doble T.

   - resultadosb1.txt → Archivo con los valores del ancho del canto de tramo de viga en caso de seleccionar sección Doble T.

Por otro lado se encuentran los archivos que almacenan valores que conciernen a los resultados de simulación de la estructura por el programa de elementos finitos, en este caso ABAQUS, estos archivos almacenarán por tanto los valores de tensiones en la estructura y desplazamiento de los nodos. Serán archivos creados por ScriptFinal.py cuando éste sea ejecutado por ABAQUS y son los siguientes:

- U2.rpt → Archivo creado por ABAQUS con información de los desplazamientos de la estructura.

- VonMisses.rpt → Archivo creado por ABAQUS con información de las tensiones de la estructura.

3.6 FUNCIONAMIENTO DE LA HERRAMIENTA.

El recorrido que realizará la herramienta por los distintos scripts y programas es el siguiente:

1. La estructura sencilla a optimizar será una viga reticulada. El usuario realizará un diseño previo introduciendo unos parámetros de entrada en la ventana de la Interfaz Gráfica, ésta se iniciará al ejecutar InterfazGraficaProyecto.m. Esos datos introducidos se guardarán en la Hoja Excel.

2. Pasamos a la iniciación de Script de AGs. En él se detallarán los parámetros de ejecución, parada y resultados de los Algoritmos Genéticos. Este script leerá los datos de la hoja Excel. Se creará la cadena de bits para un primer individuo con una longitud determinada dependiendo del tipo de sección, y ejecutará el Script Proyecto.

3. En el ScriptProyecto se descodifica la cadena de bits y se le asigna un número decimal, se crearán unos vectores que contendrán los valores de los parámetros de cada tramo de la viga que dependerán del Grado de discretización; y se guardarán en archivos temporales.

4. Hacemos que ABAQUS ejecute el Script Final, éste leerá por un lado todos los parámetros de entrada de la Hoja Excel y por otro lado los parámetros de salida guardados en los archivos temporales. Mediante cálculo de elementos finitos ABAQUS calcula los desplazamientos y tensiones en los nodos y los reporta guardándolos en dos archivos temporales.

5. Mientras se ejecuta el ScriptFinal.py tanto AGsProyecto.m como ScriptProyecto.m permanecen en espera hasta detectar los archivos temporales reportados por ABAQUS.

6. Cuando ScriptProyecto.m detecta los archivos con los datos de tensiones y desplazamientos los lee y aplica penalizaciones para aquellos que no cumplan las restricciones del problema, después de esto borra los archivos para que no se superpongan en el cálculo del siguiente individuo. Se calcula el peso de la Viga al que se le suman las penalizaciones. Este proceso se repite hasta generar tantos individuos como población haya.

7. Volvemos al AGsProyecto.m que comprueba que no se hayan cumplido criterios de parada del algoritmo, que de no ser así ejecutará una nueva iteración.

8. El proceso se repite hasta un máximo de iteraciones establecido lo suficientemente grande para que el valor óptimo no tenga ninguna penalización. En este punto del problema es donde nos encontramos con la mayor limitación de la herramienta, el tiempo computacional que suponía el cálculo del problema para el algoritmo. Este problema al que nos tuvimos que enfrentar se solucionó con la disminución de las variables, llamando variables a cada bit de la cadena de ceros y unos que crea el algoritmo genético. Para ello se redujo el grado de discretización a

niveles muy bajos y los valores que podían tomar los parámetros de las secciones para favorecer la convergencia de la solución. Es decir si el problema inicial que planteamos era una viga con sección tubular, planteábamos el problema con un Grado de Discretización de 200 y valores a tomar entre 1m y 10 centímetro para el radio exterior y entre 9 centímetros y 5 milímetros, con capacidad de optar a muchos valores intermedios la herramienta tardaba días en converger a una solución por lo que optamos por reducir estos rangos.

9. Por último una vez finalizado el cálculo el script AGsProyecto.m devuelve el resultado del peso óptimo y guarda los parámetros de las secciones.

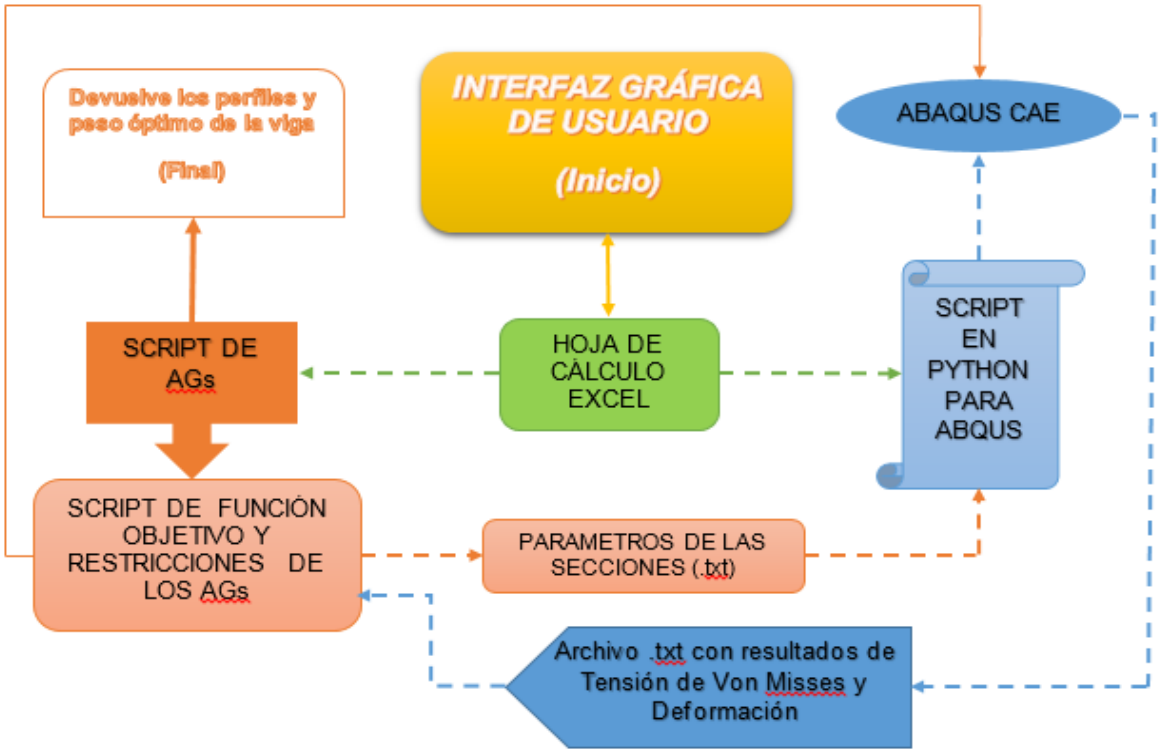


Figura 3.6.1: Esquema de funcionamiento.

4 DESARROLLO Y FUNCIONAMIENTO DE LOS DISTINTOS SCRIPTS

4.1 INTERFAZ GRÁFICA DE USUARIO

El usuario de la herramienta tan sólo trabajará con la Interfaz Gráfica donde diseñará la viga que desea optimizar. A continuación se procede a explicar el entorno de MATLAB donde se desarrolla la Interfaz Gráfica, los componentes y su funcionamiento.

La interfaz gráfica se ha creado para favorecer la manipulación de la herramienta y su elegancia ya que los parámetros necesarios para diseñar y optimizar la estructura se podrían haber introducidos en la hoja Excel, por lo que este apartado puede ser, quizás, el menos importante pero no por ello carece de explicación.

Cuando creamos una nueva interfaz gráfica en Matlab automáticamente se nos crean dos archivos, uno con extensión .fig y otro con extensión .m. El entorno donde se crea la interfaz será el archivo con extensión .fig, mientas que el archivo con extensión .m será el script donde se registran las acciones realizadas y donde podremos programar que funciones realizar con los componentes introducidos en el archivo .fig.

El entorno donde se desarrolla la interfaz es una ventana vacía donde se pueden realizar multitud de operaciones, desde cambiar el fondo a indexar una serie de componentes que tenemos disponibles en un menú, como botones, gráficas, menús emergentes, tablas…

4.1.1 DESARROLLO DE LA GUI

El archivo “InterfazGraficaProyecto.m” que se crea automáticamente tendrá la siguiente estructura:

```
function varargout = InterfazGrafica(varargin)
% INTERFAZGRAFICA MATLAB code for InterfazGrafica.fig
%      INTERFAZGRAFICA, by itself, creates a new INTERFAZGRAFICA or raises the
existing
%      singleton*.
%
%      H = INTERFAZGRAFICA returns the handle to a new INTERFAZGRAFICA or the
handle to
%      the existing singleton*.
%
%      INTERFAZGRAFICA('CALLBACK',hObject,eventData,handles,...) calls the lo-
cal
%      function named CALLBACK in INTERFAZGRAFICA.M with the given input argu-
ments.
%
%      INTERFAZGRAFICA('Property','Value',...) creates a new INTERFAZGRAFICA
or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before InterfazGrafica_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to InterfazGrafica_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help InterfazGrafica

% Last Modified by GUIDE v2.5 29-Jan-2015 10:29:46

% Begin initialization code - DO NOT EDIT
```

```
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @InterfazGrafica_OpeningFcn, ...
                   'gui_OutputFcn',  @InterfazGrafica_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT
```

```
% --- Executes just before InterfazGrafica is made visible.
function InterfazGrafica_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to InterfazGrafica (see VARARGIN)

% Choose default command line output for InterfazGrafica
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes InterfazGrafica wait for user response (see UIRESUME)
% uiwait(handles.figure1);
```

```
% --- Outputs from this function are returned to the command line.
function varargout = InterfazGrafica_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;
```

A continuación todas las tareas que realicemos en el archivo de extensión .fig quedarán reflejadas en el código. Empezaremos por cambiar la apariencia de la interfaz haciendo doble click sobre el botón izquierdo donde se despliega el menú de opciones.

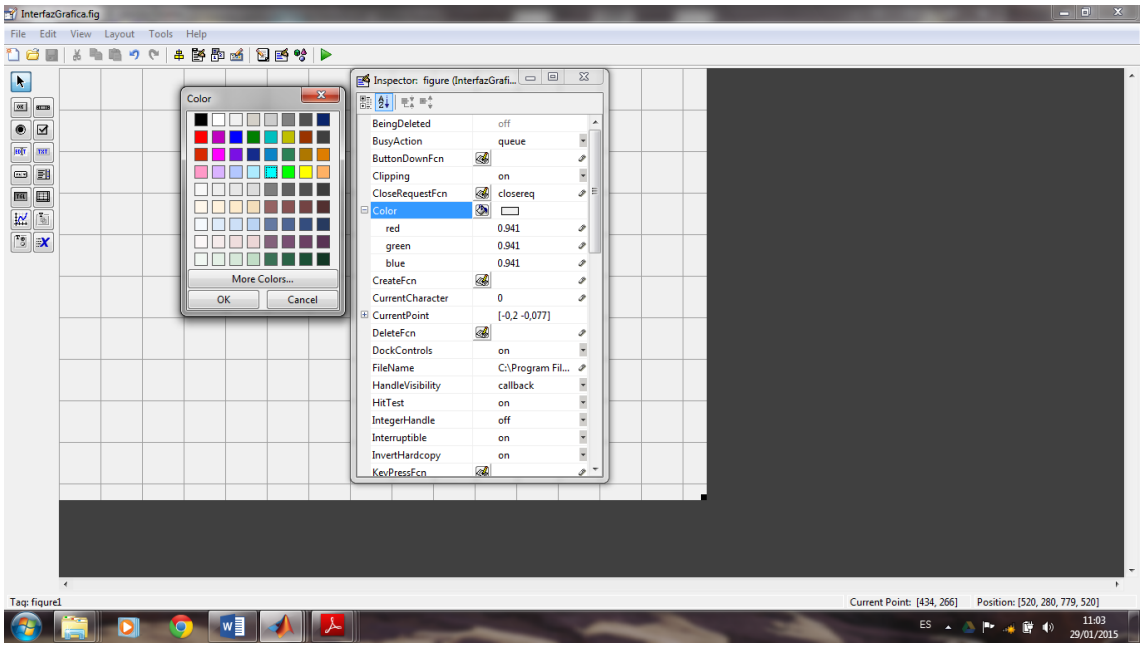


Figura 4.1.1.1: Menú de opciones de la GUI.

4.1.1.1 Implementación de "axes" para insertar Imágenes

Se procede ahora a insertar las imágenes, para ello habrá que insertar "axes", que son gráficos que admiten la inserción de imágenes.

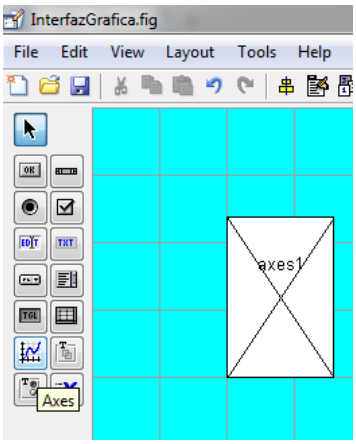


Figura 4.1.1.1.1: Creación de un axes

Insertaremos tres axes, en uno introduciremos una imagen de un puente, en otro introduciremos una donde aparecerán varios perfiles y por último introducimos un axes donde se irá apareciendo nuestra viga a medida que vayamos diseñando. Una vez insertados los dos axes pasamos al programar en el archivo con extensión .m.

Para identificar bien cada axes les modificaremos los nombres en el menú de opciones de axes. Los nombres serán: "Imagen", "ImagenPerfiles" y "Viga".

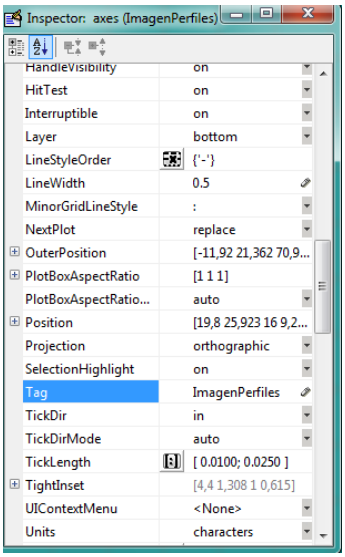


Figura 4.1.1.1.2: Menú de opciones de un axes

```
Imagen = imread('Puente4.jpg'); %Leer imagen
axes(handles.Imagen); %Carga la imagen en la grafica
axis off;          %Elimina los ejes.
imshow(Imagen); %Presenta la imagen

ImagenPerfiles = imread('Perfiles.jpg');
axes(handles.ImagenPerfiles);
axis off;
imshow(ImagenPerfiles);

axes(handles.Viga);
xlabel('Longitud de Viga');
```

4.1.1.2 Implementación de Tabla para Condiciones de Contorno

El formato que utilizaremos para las condiciones de contorno de nuestra viga, lo que en ABAQUS corresponderá al módulo LOAD que veremos más adelante, será una tabla. Para insertar una tabla pinchamos en el icono y arrastramos con el ratón.

Abriremos el menú de opciones de la tabla y la modificaremos el número de filas y columnas, de forma que nos quede la siguiente tabla editable.

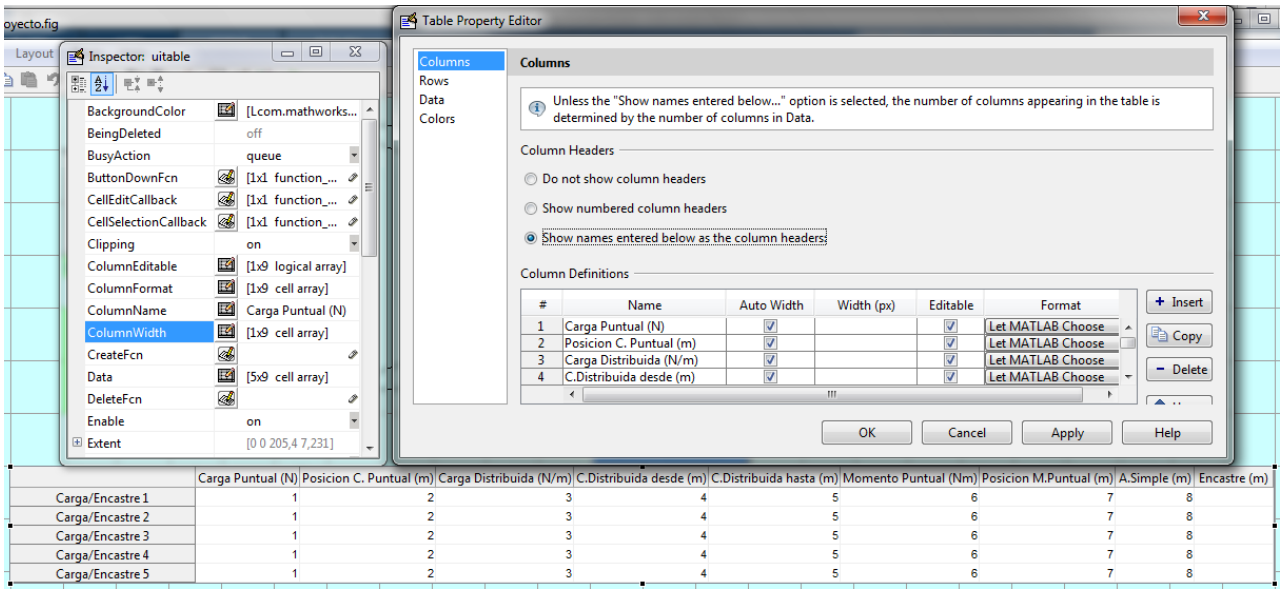


Figura 4.1.1.2.1: Menú de opciones de la Tabla

En el archivo de extensión .m se nos crean las siguientes líneas:

```
function TablaDatos_CellSelectionCallback(hObject, eventdata, handles)
% hObject    handle to TablaDatos (see GCBO)
% eventdata  structure with the following fields (see UITABLE)
%   Indices: row and column indices of the cell(s) currently selecteds
% handles    structure with handles and user data (see GUIDATA)
```

A la tabla le introducimos unos valores que quedarán como predeterminados con la finalidad de poder editarlos posteriormente y hacer referencia a ellos para programar y que la interfaz funcione correctamente.

```
a=[0 0 0 0 0 0 0 NaN NaN; 0 0 0 0 0 0 0 NaN NaN;0 0 0 0 0 0 0 NaN NaN
    ;0 0 0 0 0 0 0 NaN NaN;0 0 0 0 0 0 0 NaN NaN];

set(handles.TablaDatos,'data',a);
```

La programación del uso de los valores de la tabla y su representación en el “axes” se explicará al final.

4.1.1.3 Implementación de "Pop Menú" para la Elección de un Perfil.

Se inserta un "pop menú" y en el menú de opciones, en la pestaña String insertamos las opciones que deseamos que aparezcan cuando la ejecutemos. Se cambia también el estilo de letra y el color de fondo.

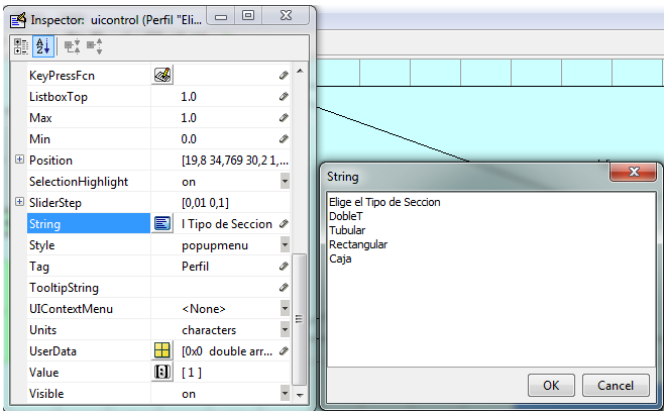


Figura 4.1.1.2.2: Menú de opciones de un "Pop Menú".

En el archivo con extensión .m se programa las acciones que se realizan dependiendo de la opción seleccionada.

```
% --- Executes on selection change in Perfil.
function Perfil_Callback(hObject, eventdata, handles)
% hObject    handle to Perfil (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns Perfil contents as
cell array
%        contents{get(hObject,'Value')} returns selected item from Perfil

val=get(hObject,'Value');
str=get(hObject,'String');
A={'DobleT','Tubular','Rectangular','Caja'};

switch str{val}
    case 'DobleT'
        xlswrite('C:\Temp\ExcelProyecto.xls',A(1),'HojaParametros','C10');
        ImagenPerfiles = imread('ImagenDobleT.jpg');
        axes(handles.ImagenPerfiles);
        axis off;
        imshow(ImagenPerfiles);
    case 'Tubular'
        xlswrite('C:\Temp\ExcelProyecto.xls',A(2),'HojaParametros','C10');
        ImagenPerfiles = imread('ImagenTubular.jpg');
        axes(handles.ImagenPerfiles);
        axis off;
        imshow(ImagenPerfiles);
    case 'Rectangular'
        xlswrite('C:\Temp\ExcelProyecto.xls',A(3),'HojaParametros','C10');
        ImagenPerfiles = imread('ImagenRectangular.jpg');
        axes(handles.ImagenPerfiles);
        axis off;
        imshow(ImagenPerfiles);
    case 'Caja'
        xlswrite('C:\Temp\ExcelProyecto.xls',A(4),'HojaParametros','C10');
        ImagenPerfiles = imread('ImagenCaja.jpg');
        axes(handles.ImagenPerfiles);
        axis off;
        imshow(ImagenPerfiles);
end

% --- Executes during object creation, after setting all properties.
function Perfil_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Perfil (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white Viga on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

El comando `switch str{val}` determina cual será el perfil seleccionado. Como se puede apreciar en el código los comandos empleados para cada caso son los mismos pero con pequeñas salvedades.

El comando `xlswrite` se utiliza para escribir en la Hoja Excel el perfil seleccionado, indicándole la ruta donde se encuentra guardado, la hoja y la celda. Será la misma celda para todos los casos.

Las siguientes tres líneas de programa cambian la imagen donde aparecían varios perfiles por una imagen del perfil seleccionado.

4.1.1.4 Implementación de "Edit Text" y "Static Text" para Introducción de los Parámetros Generales

Los Parámetros Generales son los siguientes:

- Longitud
- Grado de Discretización
- Densidad
- Módulo de Young
- Coeficiente de Poisson
- Tipo de Perfil
- Límite Elástico

De estos parámetros el Tipo de Perfil ya lo hemos definido anteriormente, ya que a tratarse de una variable con tan sólo cuatro posibles opciones hemos decidido utilizar "Pop Menú".

Para insertarlos en nuestra Interfaz Gráfica hemos empleado un "Static Text" para identificar el Parámetro y al lado del cuadro un "Edit Text" vacío para insertar el valor. Con cada uno de los valores se han programado las siguientes consignas a llevar a cabo.

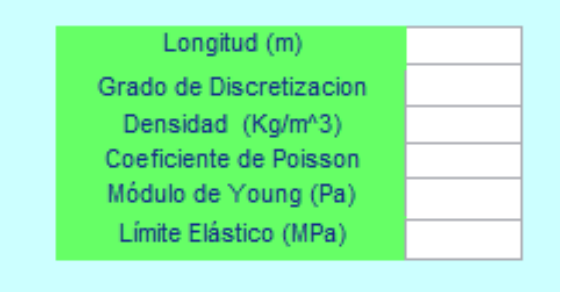


Figura 4.1.1.4.1:: Static Text y Edit Text creados.

4.1.1.4.1 Longitud

```
function Longitud_Callback(hObject, eventdata, handles)
% hObject    handle to Longitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Longitud as text
%        str2double(get(hObject,'String')) returns contents of Longitud as a
double
L=str2num(get(handles.Longitud,'string'));
xlswrite('C:\Temp\ExcelProyecto.xls',L,'HojaParametros','C5');
handles.Longitud=L;

axes(handles.Viga);
x=0:0.001:L;
handles.x=x;
handles.h=plot(x,1,'b.','linewidth',0.1);
axis([-1,L+1,0,2]);
xlabel('Longitud de Viga');

% --- Executes during object creation, after setting all properties.
function Longitud_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Longitud (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white Viga on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Primeramente leemos del “EditText” el valor que el usuario ha introducido (`L=str2num(get(handles.Longitud,'string'))`), posteriormente al igual que realizamos con el Tipo de Perfil guardamos el valor en la Hoja Excel (`xlswrite('C:\Temp\ExcelProyecto.xls', L,'HojaParametros','C5')`). Por último representamos la viga en el “axes” que habíamos creado anteriormente al que habíamos llamado “Viga”. Con el valor de la Longitud se crea una línea continua (`handles.h=plot(x,1,'b.','linewidth',0.1)`) y centramos el gráfico (`axis([-1,L+1,0,2]);`).

4.1.1.4.2 Grado de Discretización

```
function Gradode_Callback(hObject, eventdata, handles)
% hObject    handle to Gradode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Gradode as text
%        str2double(get(hObject,'String')) returns contents of Gradode as a
double

L=str2num(get(handles.Longitud,'string'));
handles.Longitud=L;

axes(handles.Viga);
x=0:0.001:L;
handles.x=x;
handles.h=plot(x,1,'b.','linewidth',0.1);
axis([-1,L+1,0,2]);
xlabel('Longitud de Viga');

G=str2num(get(handles.Gradode,'string'));
xlswrite('C:\Temp\ExcelProyecto.xls',G,'HojaParametros','C6');

L=xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C5');

hold on;
axes(handles.Viga);
xdos=0:L/G:L;
handles.x=xdos;
handles.h=plot(xdos,1,'r.','markersize',4.6);
axis([-1,L+1,0,2]);
hold off;

% --- Executes during object creation, after setting all properties.
function Gradode_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Gradode (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white Viga on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Llamamos Grado de Discretización al número de secciones distintas en que se divide la viga y a las que se le aplicará valores distintos (o por un casual, iguales) el Script de Algoritmos genéticos y se irán optimizando hasta alcanzar un peso óptimo de la viga.

Como es posible que se cambie el Grado de Discretización sin necesidad de cambiar la longitud en la primera parte dentro de esta función se repite la

representación de la Longitud en el “axes”. A continuación se lee el valor del “EditText” y se graba el valor en su celda correspondiente en la Hoja Excel.

Leemos el valor de la Longitud ya que en esta función no aparece registrado el valor. Utilizamos el comando `hold on` que nos permite dibujar encima de la gráfica sin reiniciarla y desaparezca lo representado anteriormente y pasamos a la representación del Grado de Discretización en el “axes”. Se representará cada una de las secciones a las que hace referencia el grado de Discretización con un punto rojo sobre la viga.

4.1.1.4.3 Densidad

```
function Densidad_Callback(hObject, eventdata, handles)
% hObject    handle to Densidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Densidad as text
%        str2double(get(hObject,'String')) returns contents of Densidad as a
double
D=str2num(get(handles.Densidad,'string'));
xlswrite('C:\Temp\ExcelProyecto.xls',D,'HojaParametros','C7');

% --- Executes during object creation, after setting all properties.
function Densidad_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Densidad (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white Viga on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Leemos el valor de la densidad introducido y lo guardamos en su correspondiente celda de la Hoja Excel para uso posterior en el Script de Restricciones y ABAQUS.

4.1.1.4.4 Coeficiente de Poisson

```
function Poisson_Callback(hObject, eventdata, handles)
% hObject    handle to Poisson (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Poisson as text
%        str2double(get(hObject,'String')) returns contents of Poisson as a
double
P=str2num(get(handles.Poisson,'string'));
xlswrite('C:\Temp\ExcelProyecto.xls',P,'HojaParametros','C9');

% --- Executes during object creation, after setting all properties.
function Poisson_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Poisson (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white Viga on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Leemos el valor del Coeficiente de Poisson introducido y lo guardamos en su correspondiente celda de la Hoja Excel para uso posterior en el Script de Restricciones y ABAQUS.

4.1.1.4.5 Módulo de Young

```
function Young_Callback(hObject, eventdata, handles)
% hObject    handle to Young (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of Young as text
%        str2double(get(hObject,'String')) returns contents of Young as a dou-
ble
Y=str2num(get(handles.Young,'string'));
xlswrite('C:\Temp\ExcelProyecto.xls',Y,'HojaParametros','C8');

% --- Executes during object creation, after setting all properties.
function Young_CreateFcn(hObject, eventdata, handles)
% hObject    handle to Young (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white Viga on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Leemos el valor del Módulo de Young introducido y lo guardamos en su correspondiente celda de la Hoja Excel para uso posterior en el Script de Restricciones y ABAQUS.

4.1.1.4.6 Límite Elástico.

```
function LimiteElastico_Callback(hObject, eventdata, handles)
% hObject    handle to LimiteElastico (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of LimiteElastico as text
%        str2double(get(hObject,'String')) returns contents of LimiteElastico
as a double

LE=str2num(get(handles.LimiteElastico,'string'));
xlswrite('C:\Temp\ExcelProyecto.xls',LE,'HojaParametros','C11');


% --- Executes during object creation, after setting all properties.
function LimiteElastico_CreateFcn(hObject, eventdata, handles)
% hObject    handle to LimiteElastico (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

Leemos el valor del Límite Elásticointroducido y lo guardamos en su correspondiente celda de la Hoja Excel para uso posterior en el Script de Restricciones y ABAQUS.

4.1.1.5 Implementación de “Push Button” para Tratamiento de los Valores Introducidos en la Tabla de Condiciones de Contorno.

Este icono invoca un evento inmediatamente, le llamaremos “Implantar Cargas” ya que tiene como idea principal representar las cargas y coacciones sobre la viga.

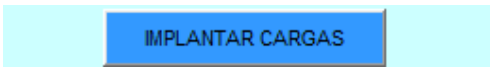


Figura 4.1.1.5.1: Push Button que ejecutará la orden de aplicar las cargas y empotramientos sobre la viga.

```
% --- Executes on button press in ImplantarCargas.
function ImplantarCargas_Callback(hObject, eventdata, handles)
% hObject    handle to ImplantarCargas (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

L=xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C5');
G=xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C6');

axes(handles.Viga);
x=0:0.001:L;
handles.x=x;
handles.h=plot(x,1,'b.','linewidth',0.1);
axis([-1,L+1,0,2]);
xlabel('Longitud de Viga');


hold on;
axes(handles.Viga);
xdos=0:L/G:L;
handles.x=xdos;
handles.h=plot(xdos,1,'r.','markersize',4.6);
axis([-1,L+1,0,2]);
hold off;

hold on;

TDatos=get(handles.TablaDatos,'data');
axes(handles.Viga);
```

En este código que vemos arriba lo que se hace es repetir la representación de la viga siguiendo los pasos idénticos que en los “EditText” de Longitud y Grado de Discretización. Por último leemos los datos introducidos en la tabla. Los datos introducidos en la Tabla son:

4.1.1.5.1 Cargas Puntuales:

```
%%%%%%%%%%CARGAS PUNTUALES%%%%%%%%%%%%%%%%%%%%
ContadorPuntuales=0;

if TDatos(1,1)<0
    handles.h=plot(TDatos(1,2),1.1:0.1:1.5,'rv');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,1),'HojaParametros','AE5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,2),'HojaParametros','AF5');
    ContadorPuntuales=ContadorPuntuales+1;
elseif TDatos(1,1)>0
    handles.h=plot(TDatos(1,2),0.5:0.1:0.9,'r^');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,1),'HojaParametros','AE5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,2),'HojaParametros','AF5');
    ContadorPuntuales=ContadorPuntuales+1;
end
```

```
if TDatos(2,1)<0
    handles.h=plot(TDatos(2,2),1.1:0.1:1.5,'rv');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,1),'HojaParametros','AE6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,2),'HojaParametros','AF6');
    ContadorPuntuales=ContadorPuntuales+1;
elseif  TDatos(2,1)>0
    handles.h=plot(TDatos(2,2),0.5:0.1:0.9,'r^');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,1),'HojaParametros','AE6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,2),'HojaParametros','AF6');
    ContadorPuntuales=ContadorPuntuales+1;
End
```

```
if TDatos(3,1)<0
    handles.h=plot(TDatos(3,2),1.1:0.1:1.5,'rv');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,1),'HojaParametros','AE7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,2),'HojaParametros','AF7');
    ContadorPuntuales=ContadorPuntuales+1;
elseif TDatos(3,1)>0
    handles.h=plot(TDatos(3,2),0.5:0.1:0.9,'r^');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,1),'HojaParametros','AE7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,2),'HojaParametros','AF7');
    ContadorPuntuales=ContadorPuntuales+1;
end
```

```
if TDatos(4,1)<0
    handles.h=plot(TDatos(4,2),1.1:0.1:1.5,'rv');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,1),'HojaParametros','AE8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,2),'HojaParametros','AF8');
    ContadorPuntuales=ContadorPuntuales+1;
elseif TDatos(4,1)>0
    handles.h=plot(TDatos(4,2),0.5:0.1:0.9,'r^');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,1),'HojaParametros','AE8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,2),'HojaParametros','AF8');
    ContadorPuntuales=ContadorPuntuales+1;
end
```

```
if TDatos(5,1)<0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,1),'HojaParametros','AE9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,2),'HojaParametros','AF9');
    handles.h=plot(TDatos(5,2),1.1:0.1:1.5,'rv');
    ContadorPuntuales=ContadorPuntuales+1;
elseif TDatos(5,1)>0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,1),'HojaParametros','AE9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,2),'HojaParametros','AF9');
    handles.h=plot(TDatos(5,2),0.5:0.1:0.9,'r^');
    ContadorPuntuales=ContadorPuntuales+1;
end

xlswrite('C:\Temp\ExcelProyecto.xls',ContadorPuntuales,'HojaParametros','AB5');
```

Para las cargas puntuales en la Interfaz Gráfica nos encontramos con dos columnas donde introducir los valores, una primera columna par el valor de las cargas (en Newton) y una segunda columna para introducir la posición de las cargas (en metros).

El valor predeterminado en todas las celdas de las dos columnas será 0. La programación de las cargas realiza lo siguiente:

1. Detecta si se ha introducido una carga positiva o negativa mediante una sentencia If.

2. Representa la carga en el "axes" viga. `handles.h =plot (TDatos(Fila,Columna de la Posicion),1.1:0.1:1.5,'rv');` La representación de la Carga puntual se realiza con una línea de triángulos discontinua perpendicular a la viga. Si la carga es negativa la carga se representa en la parte de arriba de la viga con los triángulos apuntando hacia abajo mientras que si es positiva se representa por debajo de la viga con los triángulos orientados hacia arriba.

3. Escribe los valores de las cargas y su posición en la Hoja Excel.

```
xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(Fila,Columna Va-
lor),'HojaParametros','Celda');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(Fila,Columna Posi-
cion),'HojaParametros','Celda');
```

4. Mediante un contador, se cuenta en número de cargas introducidas y escribe el valor en la Hoja Excel. Este dato es importante para que el Script de ABAQUS ya que se utiliza para crear las cargas.

```
          ContadorPuntuales=ContadorPuntuales+1;
xlswri-
te('C:\Temp\ExcelProyecto.xls',ContadorPuntuales,'HojaParametros','AB5
');
```

4.1.1.5.2 Cargas Distribuidas

```
%%%%%%%%%%%%%%%%% CARGA LINEAL O DISTRIBUIDA   %%%%%%%%%%%%%%%%

ContadorDistribuidas=0;

if TDatos(1,3)<0
    handles.h=plot(TDatos(1,4):0.2:TDatos(1,5),1.07,'gv','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,3),'HojaParametros','AG5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,4),'HojaParametros','AH5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,5),'HojaParametros','AI5');
    ContadorDistribuidas=ContadorDistribuidas+1;
elseif TDatos(1,3)>0
    handles.h=plot(TDatos(1,4):0.2:TDatos(1,5),0.93,'g^','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,3),'HojaParametros','AG5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,4),'HojaParametros','AH5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,5),'HojaParametros','AI5');
    ContadorDistribuidas=ContadorDistribuidas+1;
end
```

```
if TDatos(2,3)<0
    handles.h=plot(TDatos(2,4):0.2:TDatos(2,5),1.07,'gv','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,3),'HojaParametros','AG6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,4),'HojaParametros','AH6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,5),'HojaParametros','AI6');
    ContadorDistribuidas=ContadorDistribuidas+1;
elseif TDatos(2,3)>0
    handles.h=plot(TDatos(2,4):0.2:TDatos(2,5),0.93,'g^','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,3),'HojaParametros','AG6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,4),'HojaParametros','AH6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,5),'HojaParametros','AI6');
    ContadorDistribuidas=ContadorDistribuidas+1;
end
```

```
if TDatos(3,3)<0
    handles.h=plot(TDatos(3,4):0.2:TDatos(3,5),1.07,'gv','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,3),'HojaParametros','AG7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,4),'HojaParametros','AH7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,5),'HojaParametros','AI7');
    ContadorDistribuidas=ContadorDistribuidas+1;
elseif TDatos(3,3)>0
    handles.h=plot(TDatos(3,4):0.2:TDatos(3,5),0.93,'g^','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,3),'HojaParametros','AG7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,4),'HojaParametros','AH7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,5),'HojaParametros','AI7');
    ContadorDistribuidas=ContadorDistribuidas+1;
end
```

```
if TDatos(4,3)<0
    handles.h=plot(TDatos(4,4):0.2:TDatos(4,5),1.07,'gv','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,3),'HojaParametros','AG8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,4),'HojaParametros','AH8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,5),'HojaParametros','AI8');
    ContadorDistribuidas=ContadorDistribuidas+1;
elseif TDatos(4,3)>0
    handles.h=plot(TDatos(4,4):0.2:TDatos(4,5),0.93,'g^','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,3),'HojaParametros','AG8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,4),'HojaParametros','AH8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,5),'HojaParametros','AI8');
    ContadorDistribuidas=ContadorDistribuidas+1;
end
```

```
if TDatos(5,3)<0
    handles.h=plot(TDatos(5,4):0.2:TDatos(5,5),1.07,'gv','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,3),'HojaParametros','AG9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,4),'HojaParametros','AH9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,5),'HojaParametros','AI9');
    ContadorDistribuidas=ContadorDistribuidas+1;
elseif TDatos(5,3)>0
    handles.h=plot(TDatos(5,4):0.2:TDatos(5,5),0.93,'g^','linewidth',3);
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,3),'HojaParametros','AG9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,4),'HojaParametros','AH9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,5),'HojaParametros','AI9');
    ContadorDistribuidas=ContadorDistribuidas+1;
end

xlswri-
te('C:\Temp\ExcelProyecto.xls',ContadorDistribuidas,'HojaParametros','AB6');
```

Para las cargas distribuidas la programación es muy similar que para las Cargas lineales con la excepción de que ahora nos encontramos con que tenemos que definir la posición inicial y la posición final. En la Interfaz Gráfica nos encontramos con tres columnas donde introducir los valores, una primera columna par el valor de las cargas

(en Newton), una segunda columna para introducir la posición inicial de las cargas (en metros) y una tercera para introducir las posición final de las cargas.

El valor predeterminado en todas las celdas de las dos columnas será 0. La programación de las cargas realiza lo siguiente:

1. Detecta si se ha introducido una carga positiva o negativa mediante una sentencia If.

2. Representa la carga en el "axes" viga. `handles.h = plot(TDatos (Fila,Columna Posicion Inicial):0.2:TDatos(Fila, columnas posición Final), 1.07, 'gv', 'linewidth', 3);` La representación de la Carga Distribuida se realiza con una línea de triángulos discontinua paralela a la viga que va desde la longitud inicial a la longitud final. Si la carga es negativa la carga se representa en la parte de arriba de la viga con los triángulos apuntando hacia abajo mientras que si es positiva se representa por debajo de la viga con los triángulos orientados hacia arriba.

3. Escribe los valores de las cargas y su posición en la Hoja Excel.

```
    xlswrite('C:\Temp\ExcelProyecto.xls', TDatos(Fila, Columna Valor),'HojaParametros','Celda');
    xlswrite('C:\Temp\ExcelProyecto.xls', TDatos(Fila, Columna Posición Inicial),'HojaParametros','Celda');
    xlswrite ('C:\Temp\ExcelProyecto.xls', TDatos(Fila, Columna Posición Final),'HojaParametros','Celda');
```

4. Mediante un contador, se cuenta en número de cargas introducidas y escribe el valor en la Hoja Excel. Este dato es importante para que el Script de ABAQUS ya que se utiliza para crear las cargas.

```
ContadorDistribuidas=ContadorDistribuidas+1;

xlswrite('C:\Temp\ExcelProyecto.xls',ContadorDistribuidas,'HojaParametros','AB6');
```

4.1.1.5.3 Momento Puntual

```
ContadorMomentos=0;

if TDatos(1,6)>0
    handles.h=plot(TDatos(1,7),1.05,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,6),'HojaParametros','AJ5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,7),'HojaParametros','AK5');
    ContadorMomentos=ContadorMomentos+1;
elseif TDatos(1,6)<0
    handles.h=plot(TDatos(1,7),0.95,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,6),'HojaParametros','AJ5');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,7),'HojaParametros','AK5');
    ContadorMomentos=ContadorMomentos+1;
end
```

```
if TDatos(2,6)>0
    handles.h=plot(TDatos(2,7),1.05,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,6),'HojaParametros','AJ6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,7),'HojaParametros','AK6');
    ContadorMomentos=ContadorMomentos+1;
elseif TDatos(2,6)<0
    handles.h=plot(TDatos(2,7),0.95,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,6),'HojaParametros','AJ6');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,7),'HojaParametros','AK6');
    ContadorMomentos=ContadorMomentos+1;
end
```

```
if TDatos(3,6)>0
    handles.h=plot(TDatos(3,7),1.05,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,6),'HojaParametros','AJ7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,7),'HojaParametros','AK7');
    ContadorMomentos=ContadorMomentos+1;
elseif TDatos(3,6)<0
    handles.h=plot(TDatos(3,7),0.95,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,6),'HojaParametros','AJ7');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,7),'HojaParametros','AK7');
    ContadorMomentos=ContadorMomentos+1;
end
```

```
if TDatos(4,6)>0
    handles.h=plot(TDatos(4,7),1.05,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,6),'HojaParametros','AJ8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,7),'HojaParametros','AK8');
    ContadorMomentos=ContadorMomentos+1;
elseif TDatos(4,6)<0
    handles.h=plot(TDatos(4,7),0.95,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,6),'HojaParametros','AJ8');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,7),'HojaParametros','AK8');
    ContadorMomentos=ContadorMomentos+1;
end
```

```
if TDatos(5,6)>0
    handles.h=plot(TDatos(5,7),1.05,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,6),'HojaParametros','AJ9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,7),'HojaParametros','AK9');
    ContadorMomentos=ContadorMomentos+1;
elseif TDatos(5,6)<0
    handles.h=plot(TDatos(5,7),0.95,'mo');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,6),'HojaParametros','AJ9');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,7),'HojaParametros','AK9');
    ContadorMomentos=ContadorMomentos+1;
end


xlswrite('C:\Temp\ExcelProyecto.xls',ContadorMomentos,'HojaParametros','AB7');
```

La programación de momentos puntuales es idéntica a la de Cargas Puntuales. En la Interfaz Gráfica nos encontramos con dos columnas donde introducir los valores, una primera columna par el valor de las cargas (en Newton) y una segunda columna para introducir la posición de las cargas (en metros).

El valor predeterminado en todas las celdas de las dos columnas será 0. La programación de las cargas realiza lo siguiente:

1. Detecta si se ha introducido una carga positiva o negativa mediante una sentencia If.

2. Representa la carga en el "axes" viga. `handles.h = plot(TDatos(5,7), 0.95, 'mo');` La representación de los momentos consiste en un círculo en la posición donde se aplica. Si es el valor introducido es negativo el momento será antihorario y el círculo aparecerá un poco desplazado hacia arriba respecto a la horizontal de la viga, mientras que si el valor es positivo se representará desplazado hacia abajo.

3. Escribe los valores de las cargas y su posición en la Hoja Excel.

```
xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(Fila,Columna Va-
lor),'HojaParametros','Celda');
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(Fila,Columna Posi-
cion),'HojaParametros','Celda');
```

4. Mediante un contador, se cuenta en número de cargas introducidas y escribe el valor en la Hoja Excel. Este dato es importante para que el Script de ABAQUS ya que se utiliza para crear las cargas.

```
ContadorMomentos=ContadorMomentos+1;

xlswri-
te('C:\Temp\ExcelProyecto.xls',ContadorPuntuales,'HojaParametros','AB5
');
```

4.1.1.5.4 Apoyos Simples y Empotramientos.

```
%%%%%%%%%%%%%%%APOYOS SIMPLES%%%%%%%%%%%%%

ContadorSimples=0;


if TDatos(1,8)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,8),'HojaParametros','AM5');
    handles.h=plot(TDatos(1,8),0.92,'k^','linewidth',3);
    ContadorSimples=ContadorSimples+1;
end

if TDatos(2,8)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,8),'HojaParametros','AM6');
    handles.h=plot(TDatos(2,8),0.92,'k^','linewidth',3);
    ContadorSimples=ContadorSimples+1;
end

if TDatos(3,8)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,8),'HojaParametros','AM7');
    handles.h=plot(TDatos(3,8),0.92,'k^','linewidth',3);
    ContadorSimples=ContadorSimples+1;
end

if TDatos(4,8)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,8),'HojaParametros','AM8');
    handles.h=plot(TDatos(4,8),0.92,'k^','linewidth',3);
    ContadorSimples=ContadorSimples+1;
end

if TDatos(5,8)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,8),'HojaParametros','AM9');
    handles.h=plot(TDatos(5,8),0.92,'k^','linewidth',3);
    ContadorSimples=ContadorSimples+1;
end

xlswrite('C:\Temp\ExcelProyecto.xls',ContadorSimples,'HojaParametros','AB8');
```

```
%%%%%%%%%%%%%%% ENCASTRES    %%%%%%%%%%%

ContadorEncastres=0;

if TDatos(1,9)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(1,9),'HojaParametros','AN5');
    handles.h=plot(TDatos(1,9),0.92,'ms','linewidth',4);
    ContadorEncastres=ContadorEncastres+1;
end

if TDatos(2,9)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(2,9),'HojaParametros','AN6');
    handles.h=plot(TDatos(2,9),0.92,'ms','linewidth',4);
    ContadorEncastres=ContadorEncastres+1;
end

if TDatos(3,9)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(3,9),'HojaParametros','AN7');
    handles.h=plot(TDatos(3,9),0.92,'ms','linewidth',4);
    ContadorEncastres=ContadorEncastres+1;
end

if TDatos(4,9)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(4,9),'HojaParametros','AN8');
    handles.h=plot(TDatos(4,9),0.92,'ms','linewidth',4);
    ContadorEncastres=ContadorEncastres+1;
end

if TDatos(5,9)>=0
    xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,9),'HojaParametros','AN9');
    handles.h=plot(TDatos(5,9),0.92,'ms','linewidth',4);
    ContadorEncastres=ContadorEncastres+1;
end

xlswrite('C:\Temp\ExcelProyecto.xls',ContadorEncastres,'HojaParametros','AB9');


axis([-1,L+1,0,2]);

hold off;
```

La programación de Apoyos Simples y Empotramientos es la más sencilla, ya que únicamente hay que designar la posición en la que los queremos situar. Para cada uno de ellos hay una columna donde introducir los valores. El valor predeterminado en todas las celdas de las dos columnas será NaN que significa que no hay ningún apoyo simple ni ningún empotramiento. La programación de las cargas realiza lo siguiente:

1. Detecta si se ha variado el valor de NaN (Not a Number)
2. Representa la carga en el "axes" viga. `handles.h = plot(TDatos(Fila,Columna), 0.92, 'ms', 'linewidth',4);` La representación de apoyos simples consiste en un triángulo grueso en la

posición que se aplica mientras que la del empotramiento es un cuadrado.

3. Escribe los valores de las cargas y su posición en la Hoja Excel.

4. `xlswrite('C:\Temp\ExcelProyecto.xls',TDatos(5,9),'HojaParametros','AN9');`

5. Mediante un contador, se cuenta en número de cargas introducidas y escribe el valor en la Hoja Excel. Este dato es importante para que el Script de ABAQUS ya que se utiliza para crear las cargas.

```
    ContadorEncastres=ContadorEncastres+1;

xlswrite('C:\Temp\ExcelProyecto.xls',ContadorEncastres,'HojaParametros','AB9');
```

4.1.1.1 Implementación de "Push Button" para Optimizar estructura.

```
% --- Executes on button press in Optimizar.
function Optimizar_Callback(hObject, eventdata, handles)
% hObject    handle to Optimizar (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: slider controls usually have a light gray background.
if isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor',[.9 .9 .9]);
end



run('AGsProyecto.m')
```

La función de este "Push Button" es la de ejecutar el Script AGsProyecto `run('AGsProyecto.m')`. El Script que ejecuta la función de algoritmos genéticos. Que se explicará en el siguiente apartado (4.2).

4.1.2 MANUAL DE USO DE LA INTERFAZ GRÁFICA

PASO 1: Abrir el despegable donde pone “Elige el tipo de Sección” y elegir la sección que se desea optimizar. Podemos elegir entre Tubular, Doble T, Caja o Rectangular. Aparecerá una imagen del tipo de sección una vez seleccionado para que el usuario se haga una idea.

PASO 2: Introducir los parámetro generales teniendo en cuenta que debemos introducir la longitud antes que el Grado de Discretización ya que la representación del segundo parámetro depende de la del primero. En este momento ya nos aparecerá representada nuestra viga en la zona central de la Interfaz.

PASO 3: Se introducen las cargas y las coacciones en la viga. En caso de equivocarnos o querer quitar una carga se debe dejar en la celda el valor predeterminado por la interfaz, “0” en el caso de las cargas y “NaN” en el caso de las coacciones. Una vez introducidos en la tabla los valores de las cargas así como su posición se debe hacer click sobre el botón “Implantar Cargas”, así el programa procederá a representarlas.

PASO 4: Pulsamos el botón “OPTIMIZAR” y dejamos a la herramienta que ejecute los algoritmos para la resolución del problema.

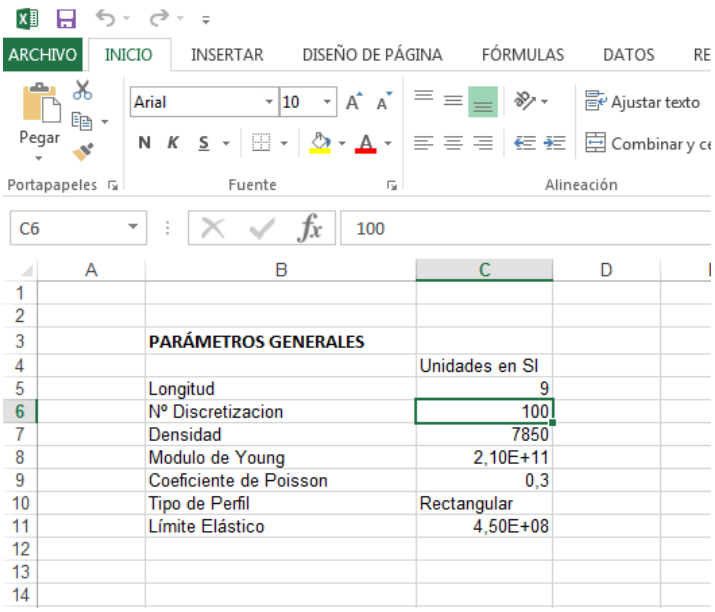
PASO 5: MATLAB nos mostrará los parámetros de cada una de las secciones así como su posición final. También quedarán guardados como archivos de extensión .txt en la carpeta C/:Temp, que es predeterminada por ABAQUS.

En el apartado 5 encontramos algunos ejemplos.

4.2 ALMACENAMIENTO Y ADMINISTRACIÓN DE DATOS: HOJA EXCEL COMO ENLACE DE INFORMACIÓN ENTRE DISTINTOS SCRIPTS Y PROGRAMAS.

Como hemos visto en el apartado anterior todas las acciones que se realizan en la Interfaz Gráfica quedan registradas en un archivo de Excel con el nombre de "ExcelProyecto.xls" y nombre de Hoja "HojaParámetros". La causa de que guardemos los datos en una Hoja de Excel es que los datos introducidos por la interfaz necesitan ser tratados posteriormente tanto por el Script de AGs como por el Script de ABAQUS. Al ser programas distintos e independientes la hoja de Excel actúa como nexo de unión entre ambos programas manteniendo constantes los valores introducidos en la interfaz.

A continuación se muestran unas imágenes de cómo se distribuyen los datos en la Hoja Excel.



	A	B	C	D
1				
2				
3		**PARÁMETROS GENERALES**		
4			Unidades en SI	
5		Longitud	9	
6		Nº Discretizacion	100	
7		Densidad	7850	
8		Modulo de Young	2,10E+11	
9		Coeficiente de Poisson	0,3	
10		Tipo de Perfil	Rectangular	
11		Límite Elástico	4,50E+08	
12				
13				
14				

Figura 3.2.1: Parámetros Generales en la Hoja de Excel

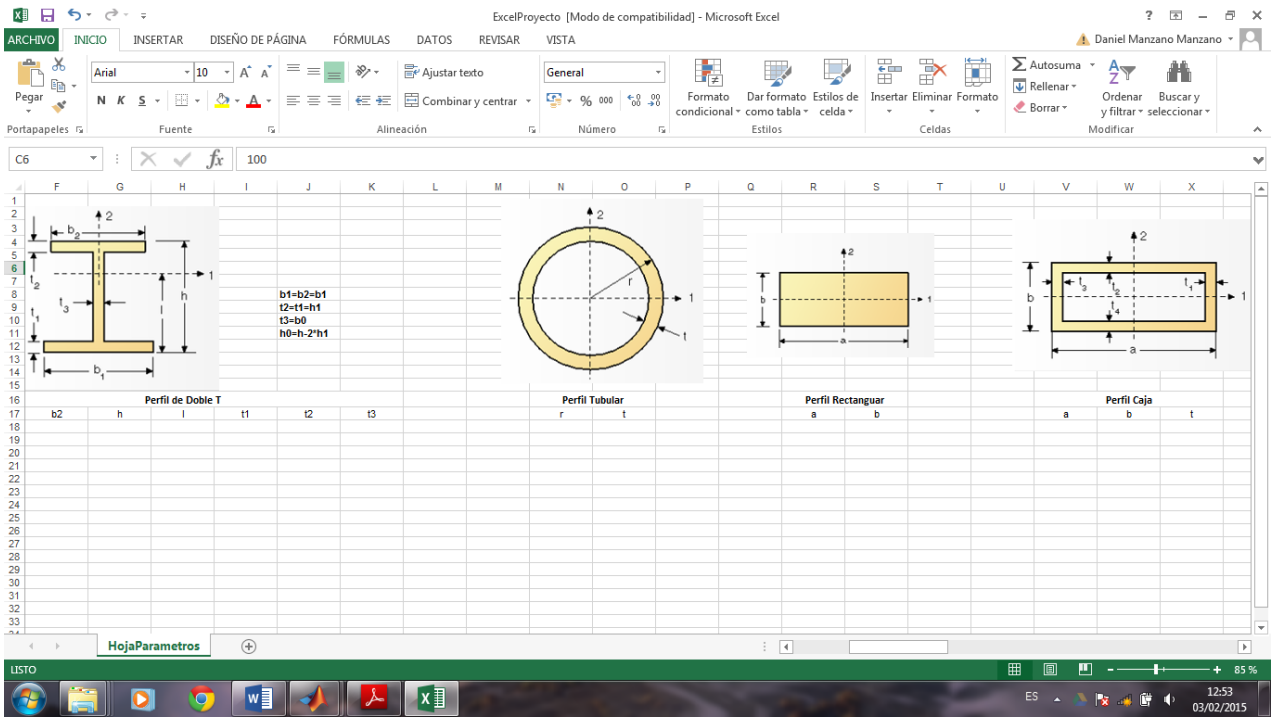


Figura 3.2.2: Hoja Excel: Tipos de Perfiles donde se reflejan los parámetros que solicita ABAQUS.

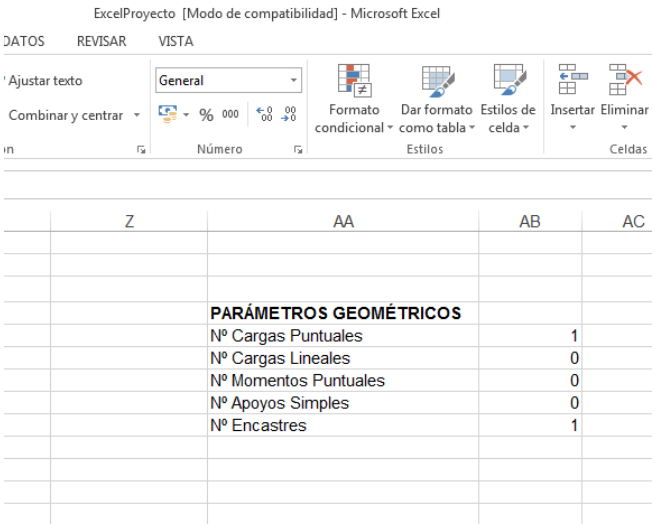
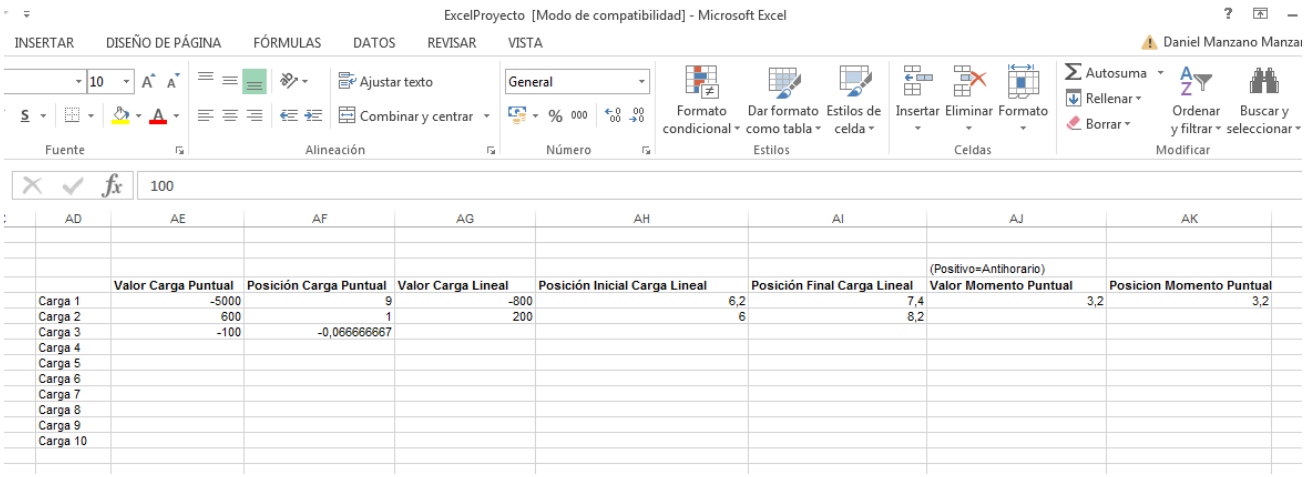
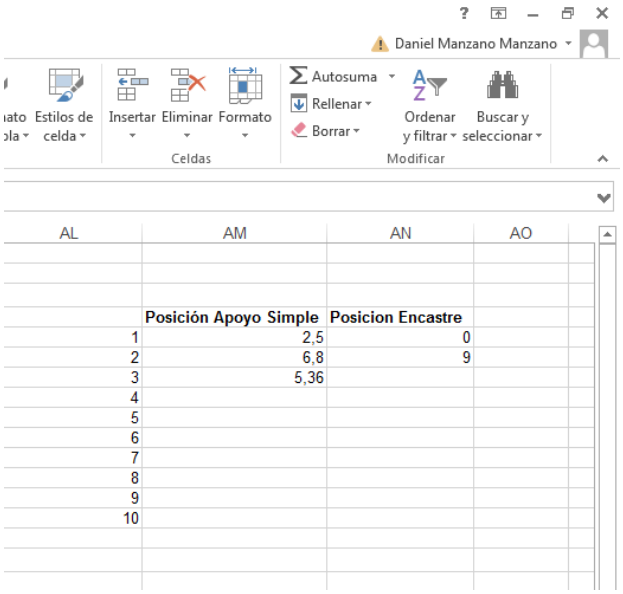


Figura 3.2.3: Parámetros Geométricos en la Hoja de Excel



	AD	AE	AF	AG	AH	AI	AJ	AK
							(Positivo=Antihorario)	
		Valor Carga Puntual	Posición Carga Puntual	Valor Carga Lineal	Posición Inicial Carga Lineal	Posición Final Carga Lineal	Valor Momento Puntual	Posicion Momento Puntual
	Carga 1	-5000	9	-800	6,2	7,4	3,2	3,2
	Carga 2	600	1	200	6	8,2		
	Carga 3	-100	-0,066666667					
	Carga 4							
	Carga 5							
	Carga 6							
	Carga 7							
	Carga 8							
	Carga 9							
	Carga 10							

Figura 3.2.1: Cargas y posición de aplicación guardados en la Hoja Excel



	AL	AM	AN	AO
		Posición Apoyo Simple	Posicion Encastre	
1		2,5	0	
2		6,8	9	
3		5,36		
4				
5				
6				
7				
8				
9				
10				

Figura 3.2.1: Posición de Apoyos simples y Empotramientos guardados en la Hoja de Excel

4.3 SCRIPT DE ALGORITMO GENÉTICO

4.3.1 INTRODUCCIÓN

Una vez diseñada la viga por el usuario con los guardados los datos en la Hoja Excel de forma automatizada se vio anteriormente que para iniciar el proceso de optimización se pulsaba el botón “OPTIMIZAR” en la Interfaz Gráfica cuya única función es ejecutar el Script de Algoritmo Genético, un archivo de MATLAB con extensión .m que recibe el nombre de “AGsProyecto.m”. La función que tiene este Script es la misma que si abriéramos y utilizásemos la Toolbox pero con la diferencia que de esta forma el proceso se realizará de manera automática con los parámetros de los algoritmos ya introducidos de forma que se le resta flexibilidad al usuario privándole de elegir los parámetros de los AGs a cambio de ahorrar trabajo y tiempo. Se probarán distintos casos y ejemplos variando los parámetros hasta encontrar una combinación con la cual se obtengan mejores resultados.

El Script de Algoritmo Genético es el encargado de Optimizar la Función Ojetivo, dicha Función Objetivo se encuentra guardada en otro Script llamado “ScriptProyecto”. Esa función Objetivo es el peso de la viga.

Aunque el “ScriptProyecto” se explica en el apartado siguiente se irán dando algunos datos acerca de su contenido y características ya que ambos trabajan de forma compenetrada.

4.3.2 DEFINICIÓN DE VARIABLES

Comenzamos a explicar la programación del Script de Algoritmos Genéticos:

Primeramente cerramos los archivos y ventanas que pudiéramos tener emergentes o abiertas y puedan interrumpir el proceso de optimización incluido la ventana de interfaz gráfica (`close all; clear all; warning('off').`

A continuación definimos las variables dentro de este Script, para ello hacemos referencia a la “HojaParámetros” de Excel donde tenemos guardados los datos. Posteriormente convertimos las Variables en Variables Globales, lo que nos permite utilizar la variable en todo momento, es decir en el archivo “ScriptProyecto” no será necesario leer de nuevo las variables del Excel, simplemente introduciremos el comando “global” seguido de la variable y ya la tendremos definida.

```
% ***************************************************
% Optimizacion Viga : Resolución mediante AAGG
% ***************************************************
close all;
clear all;
warning('off');

LongitudViga=xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C5');

Densidad=xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C7');

LimiteElastico=xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C11');

NSecciones=xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C6');

LongitudSeccion=LongitudViga/NSecciones;

[Num,Seccion] = xlsread('C:\Temp\ExcelProyecto.xls','HojaParametros','C10');

Seccion=char(Seccion);
Seccion=num2str(Seccion);

global NSecciones;
global Seccion;
global LongitudViga;
global Densidad;
global LimiteElastico;
global LongitudSeccion;
```

Mediante el comando `strcmp` que se utiliza para comparar cadenas comprobamos el tipo de sección que ha elegido el usuario. Una vez comprobada se declara la variable `Longitud` y se le asigna un valor distinto dependiendo del tipo de sección. El valor asignado hace referencia a la longitud de cadena de bits que tiene una sección. Por ejemplo, para el caso de sección rectangular tenemos dos parámetros, base y altura, si cada uno de los valores se comprenden entre 0.5 y 0.1 con una precisión de 0.1 aplicando la *Ecuación 2.1* tenemos una longitud de cadena de 6 bits para esa sección. Tres bits para la altura y otros tres para la base. Se verá con más detenimiento en el “ScriptProyecto”. En un principio se establecieron tolerancias mayores a las cadenas, es decir, cada sección podría llegar a tener hasta 18 bits, 9 para cada parámetro pero eran demasiadas variables para que el algoritmo llegara a converger.

```
if strcmp(Seccion,'Tubular')==1
    Longitud=6;
end

if strcmp(Seccion,'Rectangular')==1
    Longitud=6;
end

if strcmp(Seccion,'Caja')==1
    Longitud=9;
end

if strcmp(Seccion,'DobleT')==1
    Longitud=6;
end
```

4.3.3 PARÁMETROS DE LOS AGs

4.3.3.1 Población

```
%%%Generamos las opciones de algoritmos geneticos%%%
options = gaoptimset;

% Población Genética
options = gaoptimset(options, 'PopulationSize' , 30);
options = gaoptimset(options, 'PopulationType' ,'bitstring');
options = gaoptimset(options, 'CreationFcn',@gacreationuniform);
```

Se empieza la programación de parámetros activando la opción `options = gaoptimset;` estructura que contiene las opciones de optimización. A continuación se programan las opciones de población.

- Población Inicial (`'PopulationSize'`): Se utilizan valores alrededor de 30 debido que si incrementábamos la población aumentaba de forma considerable el tiempo computacional.
- Tipo de población (`'PopulationType'`): Será una población de bits (ceros y unos) por defecto, este valor no cambiará.
- Creación de la Población (`'CreationFcn'`): Se creará de manera uniforme `,@gacreationuniform.`

4.3.3.2 Criterios de Parada

```
% Criterios de parada
options = gaoptimset(options, 'Generations', 10);
options = gaoptimset(options, 'StallGenLimit', 50);
```

El criterio de parada estará basado principalmente en el número de generaciones del algoritmo. Si la tolerancia por defecto de los AGs no varía a lo largo de 50 generaciones consideraremos que la solución ha convergido y el programa se detendrá dando la solución óptima.

4.3.3.3 Elitismo

```
% ******************************************************
% Operadores Genéticos
% ******************************************************

% Elitismo
options = gaoptimset(options, 'EliteCount', 2);
```

El elitismo se puede interpretar como una opción de reproducción o una condición de reemplazo, especifica el número de individuos que están garantizados para sobrevivir a la siguiente generación, para los problemas. Mantendremos un grado de elitismo de 2 que es el predeterminado para problemas no enteros.

4.3.3.4 Operador de Selección

```
% Operador de selección
options = gaoptimset(options, 'SelectionFcn',@selectionroulette);
```

Tras probar con distintas opciones de selección se ha comprobado que la más adecuada para nuestro problema es la selección por ruleta si se utilizaban números de variables muy elevados, en nuestros ejemplos se ha utilizado `@selectionstochunif`, o selección estocástica uniforme.

4.3.3.5 Función de Cruce

```
% Algoritmo de cruce
options = gaoptimset(options, 'CrossoverFcn', @crossoverscattered);
options = gaoptimset(options, 'CrossoverFraction', 0.9);
```

La función de cruce elegida será `@crossoverscattered`, también seleccionada teniendo en cuenta que tanto el cruce monopunto, en dos puntos y uniforme hacía que la población se homogenizase llevándola a converger demasiado rápido sin alcanzar un buen resultado.

4.3.3.6 Algoritmo de Mutación

```
% Algoritmo de mutación
options = gaoptimset(options, 'MutationFcn',{@mutationuniform, 0.45});
```

Tendremos una mutación uniforme con un alto grado de mutación de 0.45 debido a que con un valor bajo la población tendía a converger demasiado rápido.

4.3.3.7 Configuración de Salida

```
% ******************************************************
% Configuracion Salida
% ******************************************************

options = gaoptimset(options, 'Display', 'off');
options = gaoptimset(options, 'PlotInterval', 1);
options = gaoptimset(options, 'PlotFcns',[@gaplotbestf]);
```

Con `'Display', 'off'` conseguimos que no se muestre ninguna mientras el algoritmo se está ejecutando. Utilizamos un `'PlotInterval'` con valor 1 que especifica el número de generaciones entre llamadas consecutivas a las funciones de la trama, en la gráfica se nos mostrará el mejor individuo y la media en cada generación.

4.3.4 EJECUCIÓN DEL ALGORITMO

```
% Ejecución algoritmo

[x, fval, reason, output, population, scores]= ...
   ga(@ScriptProyecto,Longitud*NSecciones,options);
```

`[x,fval,exitflag,output,population,scores]=ga(fitnessfcn,nvars,...)` devuelve resultados de las puntuaciones de la población final.

FVAL es la función de aptitud evaluada en x. Ejecuta el @ScriptProyecto, que como hemos dicho antes hace referencia a la función objetivo.

`Longitud*NSecciones` es el número de variables que tenemos en nuestro problema, es decir, el número de ceros y unos.

4.3.5 CODIFICACIÓN DEL RESULTADO

```
%%%Decodificamos la solucion que la obtenemos en binario y la pasamos
%%%a numero decimal.

if strcmp(Seccion,'Tubular')==1

    xmax=[0.5 0.049];

    xmin=[0.1 0.01];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.1 Y m=0.01

    Longitud1=3;

    Longitud2=3;

    Longitud3=0;

    %Definimos el resto de Variables

    Contador0=0;

    Contador1=1;

    rs = [];

    ts = [];
```

Empezamos con la programación para descodificar primeramente los resultados obtenidos en el caso de haber seleccionado una viga de sección tubular. Primeramente definimos los valores máximos y mínimo entre los que puede variar los parámetros de la sección.

```
xmax=[0.5 0.049];

xmin=[0.1 0.01];
```

Con estos valores y con una precisión de 0.1 para radio y de 0.01 para el espesor calculamos la longitud de la cadena que se ha creado para cada parámetro obteniendo 3 bits dedicados a cada uno de los radios y 3 bits dedicados a cada uno de los espesores. Es decir que si tenemos una viga con un grado de discretización de 30tendremos una cadena de 30x6= 180 bits.

```
Longitud1=3;

Longitud2=3;
```

El radio exterior del perfil tubular será la variable `rs`, será un vector donde se almacenarán todos los radios de cada una de las secciones, su valor podrá comprender entre 0.5 y 0.1 metros.

El espesor de la sección tubular será la variable `ts`, será un vector donde se almacenarán todos los espesores de cada una de las secciones y su valor podrá comprender entre 0.049 y 0.01 metros.

A continuación definimos variables que actuarán como contadores, y los vectores de radios y espesores vacios.

```
While Contador0<NSecciones

    %Pasamos de binario a numero real para evaluar la funcion objetivo

    c=x(Longitud2*Contador0+Longitud1*Contador0+1:Longitud1*Contador1+
        Longitud2*Contador0);
    c1=num2str(c);
    r1=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));

    v=x(Longitud1*Contador1+Longitud2*Contador0+1:Longitud1*Contador1+
        Longitud2*Contador1);
    v1=num2str(v);
    t1=xmin(2)+bin2dec(v1)*((xmax(2)-xmin(2))/(2^Longitud2-1));

    rs=[rs,r1];
    ts=[ts,t1];

    Contador0=Contador0+1;
    Contador1=Contador1+1;

end

Contador0=0;
Contador1=1;

disp('A continuación se muestran los valores de las distintas Secciones');
disp('')
disp('Los datos tambien quedaran guardados en un archivo txt')
```

Comenzamos a descodificar a número decimal el resultado final del algoritmo. `c` y `v` son a la variable a la que asignaremos los bits correspondientes a cada parámetro. `c1` y `v1` son las variables que poseen el valor en número entero de `c` y `v`. Finalmente `r1` y `t1` son las variables que poseen el valor completamente descodificado en número decimal. Mediante un bucle while rellenamos los vectores `rs` y `ts` con todos los valores de cada una de las secciones.

```
    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('Radio Exterior ='); disp(rs(i)); disp('Espesor ='); disp(ts(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosrs.txt','w+');
    fprintf(fid1,'%11.6f',rs);

    fid2=fopen('C:\temp\resultadosts.txt','w+');
    fprintf(fid2,'%11.6f',ts);

    Contador1=1;

    Area=pi*((rs).^2-(rs-ts).^2);

end
```

Una vez que hemos descodificado todos los valores de las secciones pasamos a mostrarlos en pantalla, para ello iniciamos un bucle “for” que recorre ambos parámetros. Para finalizar guardamos los datos en un archivo con extensión .txt bajo unas condiciones dadas de forma que luego podamos utilizarlo. Se guardará cada parámetro (radio y espesor) en dos archivos distintos, los valores se guardarán ocupando 11 espacios de los cuales 6 de ellos pertenecerán a la parte decimal.

Por último se calcula el área de cada sección quedando almacenados en el vector `Area`.

Para el resto de secciones se realiza el mismo proceso.

```
if strcmp(Seccion,'Rectangular')==1

    xmax=[0.5 0.5];

    xmin=[0.1 0.1];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.1

    Longitud1=3;

    Longitud2=3;

    Longitud3=0;

    %Definimos el resto de Variables

    warning('off');

    Contador0=0;

    Contador1=1;

    ar = [];

    br = [];
```

```
while Contador0<NSecciones

    %Pasamos de binario a numero real para evaluar la funcion objetivo

    c=x(Longitud2*Contador0+Longitud1*Contador0+1:Longitud1*Contador1+
        Longitud2*Contador0);
    c1=num2str(c);
    ar1=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));

    v=x(Longitud1*Contador1+Longitud2*Contador0+1:Longitud1*Contador1+
        Longitud2*Contador1);
    v1=num2str(v);
    br1=xmin(2)+bin2dec(v1)*((xmax(2)-xmin(2))/(2^Longitud2-1));


    ar=[ar,ar1];
    br=[br,br1];

    Contador0=Contador0+1;
    Contador1=Contador1+1;

end

Contador0=0;
Contador1=1;
```

```
    disp('A continuación se muestran los valores de las distintas Secciones ');
    disp('')
    disp('Los datos tambien quedaran guardados en un archivo txt');

    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('Lado1 ='); disp(ar(i)); disp('Lado2 ='); disp(br(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosar.txt','w+');
    fprintf(fid1,'%11.6f',ar);

    fid2=fopen('C:\temp\resultadosbr.txt','w+');
    fprintf(fid2,'%11.6f',br);

    for i=1:NSecciones
        Area(i)=ar(i)*br(i);
    end

end
```

Como podemos ver en el perfil rectangular tenemos dos parámetros en la sección, la base y la altura del rectángulo, `ar` y `br`, a los cuales se les asigna la misma longitud de cadena, una longitud de 3 bits para cada uno. Los valores comprenderán entre 0.5 y 0.1 con una precisión de 0.1.

```
if strcmp(Seccion,'Caja')==1

    xmax=[0.5 0.5 0.049];

    xmin=[0.1 0.1 0.01];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.1 y m=0.01

    Longitud1=3;

    Longitud2=3;

    Longitud3=3;

    %Definimos el resto de Variables

    Contador0=0;

    Contador1=1;

    ac = [];

    bc = [];

    tc = [];
```

```
while Contador0<NSecciones

        %Pasamos de binario a numero real para evaluar la funcion objetivo

        c=x(Longitud3*Contador0+Longitud2*Contador0+Longitud1*Contador0+
            1:Longitud1*Contador1+Longitud3*Contador0+Longitud2*Contador0);
        c1=num2str(c);
        ac1=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));

        c2=x(Longitud3*Contador0+Longitud1*Contador1+Longitud2*Contador0+
            1:Longitud2*Contador1+Longitud3*Contador0+Longitud1*Contador1);
        c3=num2str(c2);
        bc1=xmin(2)+bin2dec(c3)*((xmax(2)-xmin(2))/(2^Longitud2-1));

        v=x(Longitud2*Contador1+Longitud1*Contador1+Longitud3*Contador0+
            1:Longitud3*Contador1+Longitud2*Contador1+Longitud1*Contador1);
        v1=num2str(v);
        t1=xmin(3)+bin2dec(v1)*((xmax(3)-xmin(3))/(2^Longitud3-1));

        ac=[ac,ac1];
        bc=[bc,bc1];
        tc=[tc,t1];

        Contador0=Contador0+1;
        Contador1=Contador1+1;

    end

    Contador0=0;
    Contador1=1;
```

```
    disp('A continuación se muestran los valores de las distintas Secciones ');
    disp('')
    disp('Los datos tambien quedaran guardados en un archivo txt');


    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('Lado1='); disp(ac(i)); disp('Lado2='); disp(bc(i));
        disp('Espesor='); disp(tc(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosac.txt','w+');
    fprintf(fid1,'%11.6f',ac);

    fid2=fopen('C:\temp\resultadosbc.txt','w+');
    fprintf(fid2,'%11.6f',bc);

    fid3=fopen('C:\temp\resultadostc.txt','w+');
    fprintf(fid3,'%11.6f',tc);

    for i=1:NSecciones
        Area(i)=ac(i)*bc(i)-((ac(i)-tc(i)*2)*(bc(i)-tc(i)*2));
    end

end
```

Para el perfil caja nos encontramos con tres parámetros en la sección, base, algura y espesor, `ac`, `bc` y `tc`. Arriba en el código podemos ver detallado cuáles son las longitudes de cadena así como los valores máximos y mínimos que puede tomar cada cadena.

```
if strcmp(Seccion,'DobleT')==1

    xmax=[0.5 0.05];

    xmin=[0.1  0.01];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.01 y m=0.001

    Longitud1=3;

    Longitud2=3;

    Longitud3=0;

    %Definimos el resto de Variables

    Contador0=0;

    Contador1=1;

    h0 = [];

    h1 = [];

    b0 = [];

    b1 = [];
```

```
while Contador0<NSecciones

        %Pasamos de binario a numero real para evaluar la funcion objetivo


c=x(Longitud2*Contador0+Longitud1*Contador0+1:Longitud1*Contador1+Longitud2*Cont
ador0);
        c1=num2str(c);
        ht0=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));


v=x(Longitud1*Contador1+Longitud2*Contador0+1:Longitud1*Contador1+Longitud2*Cont
ador1);
        v1=num2str(v);
        ht1=xmin(2)+bin2dec(v1)*((xmax(2)-xmin(2))/(2^Longitud2-1));

        h0=[h0,ht0];
        h1=[h1,ht1];
        b0=[b0,ht0/33];
        b1=[b1,ht1*10];

        Contador0=Contador0+1;
        Contador1=Contador1+1;

    end

    Contador0=0;
    Contador1=1;
```

```
    disp('A continuación se muestran los valores de las distintas Secciones ');
    disp('')
    disp('Los datos tambien quedaran guardados en un archivo txt')

    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('h0='); disp(h0(i)); disp('h1='); disp(h1(i));
        disp('b0='); disp(b0(i)); disp('b1='); disp(b1(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosh0.txt','w+');
    fprintf(fid1,'%11.6f',h0);

    fid2=fopen('C:\temp\resultadosh1.txt','w+');
    fprintf(fid2,'%11.6f',h1);

    fid3=fopen('C:\temp\resultadosb0.txt','w+');
    fprintf(fid3,'%11.6f',b0);

    fid4=fopen('C:\temp\resultadosb1.txt','w+');
    fprintf(fid4,'%11.6f',b1);

    for i=1:NSecciones
        Area(i)=h0(i)*b0(i)+2*h1(i)*b1(i);
    end

end
```

Para el perfil en Doble T tenemos una excepción, es que tan sólo se decodifican dos variables pero declaramos cuatro, haremos que dos variables dependan de las dos anteriores como puede verse en el código.

El esquema de las secciones con sus respectivos parámetros puede verse en el apartado 3.5.1 dentro del Script de Python.

```
Volumen=Area*LongitudSeccion;
Peso=Volumen*Densidad;
PesoTotal=0;

for i=1:NSecciones
    PesoTotal=PesoTotal+Peso(i);
end

disp('El peso total de la viga es:'); disp(PesoTotal); disp('Kg');
```

Para terminar con el Script de AGs se calcula el peso total y se muestra en pantalla.

4.4 SCRIPT DE FUNCIÓN OBJETIVO Y RESTRICCIONES

En esté Script se definirá la función objetivo de los AGs y se le aplicarán las restricciones que debe cumplir para llegar al valor óptimo. La función objetivo será el Peso de la viga. Este Script se realiza en MATLAB y se guardará el archivo con una extensión .m con el nombre de “ScritpProyecto.m”.

4.4.1 DECLARACIÓN DE FUNCIÓN OBJETIVO Y DEFINICIÓN DE VARIABLES

```
function y=PesoViga(x)

%PerfilTubular

warning('off');
global Seccion;
global NSecciones;
global Densidad;
global LimiteElastico;
global LongitudSeccion;
```

Se declara la función a optimizar `function y=PesoViga(x)` y a continuación aplicamos el comando global a las variables que necesitamos utilizar en el Script y que se había definido con anterioridad en el Script “AGsProyecto”.

A continuación se descodifican las variables y se pasan a número decimal de la misma forma que en el Script “AGsProyecto”.Una vez descodificadas se almacenan en sus respectivos vectores y los valores de estos vectores se guardan en un archivo con extensión .txt.

```
if strcmp(Seccion,'Tubular')==1

    xmax=[0.5 0.05];

    xmin=[0.1 0.01];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.1 Y M=0.01

    Longitud1=3;

    Longitud2=3;

    Longitud3=0;

    %Definimos el resto de Variables

    Contador0=0;

    Contador1=1;

    rs = [];

    ts = [];
```

```
    While Contador0<NSecciones

        %Pasamos de binario a numero real para evaluar la funcion objetivo

        c=x(Longitud2*Contador0+Longitud1*Contador0+1:Longitud1*Contador1+
            Longitud2*Contador0);
        c1=num2str(c);
        r1=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));

        v=x(Longitud1*Contador1+Longitud2*Contador0+1:Longitud1*Contador1+
            Longitud2*Contador1);
        v1=num2str(v);
        t1=xmin(2)+bin2dec(v1)*((xmax(2)-xmin(2))/(2^Longitud2-1));

        rs=[rs,r1];
        ts=[ts,t1];

        Contador0=Contador0+1;
        Contador1=Contador1+1;

    end

    Contador0=0;
    Contador1=1;

    disp('A continuación se muestran los valores de las distintas Secciones');
    disp('')
    disp('Los datos tambien quedaran guardados en un archivo txt')
```

```
    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('Radio Exterior ='); disp(rs(i)); disp('Espesor ='); disp(ts(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosrs.txt','w+');
    fprintf(fid1,'%11.6f',rs);

    fid2=fopen('C:\temp\resultadosts.txt','w+');
    fprintf(fid2,'%11.6f',ts);

    Contador1=1;

end
```

```
if strcmp(Seccion,'Rectangular')==1

    xmax=[0.5 0.5];

    xmin=[0.1 0.1];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.1

    Longitud1=3;

    Longitud2=3;

    Longitud3=0;

    %Definimos el resto de Variables

    Contador0=0;

    Contador1=1;

    ar = [];

    br = [];
```

```
    while Contador0<NSecciones

        %Pasamos de binario a numero real para evaluar la funcion objetivo

        c=x(Longitud2*Contador0+Longitud1*Contador0+1:Longitud1*Contador1+
            Longitud2*Contador0);
        c1=num2str(c);
        ar1=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));

        v=x(Longitud1*Contador1+Longitud2*Contador0+1:Longitud1*Contador1+
            Longitud2*Contador1);
        v1=num2str(v);
        br1=xmin(2)+bin2dec(v1)*((xmax(2)-xmin(2))/(2^Longitud2-1));


        ar=[ar,ar1];
        br=[br,br1];

        Contador0=Contador0+1;
        Contador1=Contador1+1;

    end

    Contador0=0;
    Contador1=1;
```

```
    disp('A continuación se muestran los valores de las distintas Secciones ');
    disp('')
    disp('Los datos tambien quedaran guardados en un archivo txt');

    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('Lado1 ='); disp(ar(i)); disp('Lado2 ='); disp(br(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosar.txt','w+');
    fprintf(fid1,'%11.6f',ar);

    fid2=fopen('C:\temp\resultadosbr.txt','w+');
    fprintf(fid2,'%11.6f',br);

end
```

```
if strcmp(Seccion,'Caja')==1

    xmax=[0.5 0.5 0.049];

    xmin=[0.1 0.1 0.01];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.1 y m=0.01

    Longitud1=3;

    Longitud2=3;

    Longitud3=3;

    %Definimos el resto de Variables

    Contador0=0;

    Contador1=1;

    ac = [];

    bc = [];

    tc = [];
```

```
while Contador0<NSecciones

        %Pasamos de binario a numero real para evaluar la funcion objetivo

        c=x(Longitud3*Contador0+Longitud2*Contador0+Longitud1*Contador0+
            1:Longitud1*Contador1+Longitud3*Contador0+Longitud2*Contador0);
        c1=num2str(c);
        ac1=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));

        c2=x(Longitud3*Contador0+Longitud1*Contador1+Longitud2*Contador0+
            1:Longitud2*Contador1+Longitud3*Contador0+Longitud1*Contador1);
        c3=num2str(c2);
        bc1=xmin(2)+bin2dec(c3)*((xmax(2)-xmin(2))/(2^Longitud2-1));

        v=x(Longitud2*Contador1+Longitud1*Contador1+Longitud3*Contador0+
            1:Longitud3*Contador1+Longitud2*Contador1+Longitud1*Contador1);
        v1=num2str(v);
        t1=xmin(3)+bin2dec(v1)*((xmax(3)-xmin(3))/(2^Longitud3-1));

        ac=[ac,ac1];
        bc=[bc,bc1];
        tc=[tc,t1];

        Contador0=Contador0+1;
        Contador1=Contador1+1;

    end

    Contador0=0;
    Contador1=1;
```

```
    disp('A continuación se muestran los valores de las distintas Secciones ');
    disp('')
    disp('Los datos tambien quedaran guardados en un archivo txt');


    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('Lado1='); disp(ac(i)); disp('Lado2='); disp(bc(i));
        disp('Espesor='); disp(tc(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosac.txt','w+');
    fprintf(fid1,'%11.6f',ac);

    fid2=fopen('C:\temp\resultadosbc.txt','w+');
    fprintf(fid2,'%11.6f',bc);

    fid3=fopen('C:\temp\resultadostc.txt','w+');
    fprintf(fid3,'%11.6f',tc);

end
```

```
if strcmp(Seccion,'DobleT')==1

    xmax=[0.5 0.05];

    xmin=[0.1  0.01];

    %Se calcula la longitud de cada variable teniendo en cuenta la máxima y
    %mínima con precisión m=0.1 y m=0.01

    Longitud1=3;

    Longitud2=3;

    Longitud3=0;

    %Definimos el resto de Variables

    Contador0=0;

    Contador1=1;

    h0 = [];

    h1 = [];

    b0 = [];

    b1 = [];
```

```
while Contador0<NSecciones

        %Pasamos de binario a numero real para evaluar la funcion objetivo


c=x(Longitud2*Contador0+Longitud1*Contador0+1:Longitud1*Contador1+Longitud2*Cont
ador0);
        c1=num2str(c);
        ht0=xmin(1)+bin2dec(c1)*((xmax(1)-xmin(1))/(2^Longitud1-1));


v=x(Longitud1*Contador1+Longitud2*Contador0+1:Longitud1*Contador1+Longitud2*Cont
ador1);
        v1=num2str(v);
        ht1=xmin(2)+bin2dec(v1)*((xmax(2)-xmin(2))/(2^Longitud2-1));

        h0=[h0,ht0];
        h1=[h1,ht1];
        b0=[b0,ht0/33];
        b1=[b1,ht1*10];

        Contador0=Contador0+1;
        Contador1=Contador1+1;

    end

    Contador0=0;
    Contador1=1;
```

```
    disp('A continuación se muestran los valores de las distintas Secciones ');
    disp('')
    disp('Los datos tambien quedaran guardados en un archivo txt')

    for i=1:NSecciones
        N='Seccion';
        N=[N int2str(Contador1)];
        disp(N);
        disp('h0='); disp(h0(i)); disp('h1='); disp(h1(i));
        disp('b0='); disp(b0(i)); disp('b1='); disp(b1(i));
        Contador1=Contador1+1;
    end

    Contador1=1;

    fid1=fopen('C:\temp\resultadosh0.txt','w+');
    fprintf(fid1,'%11.6f',h0);

    fid2=fopen('C:\temp\resultadosh1.txt','w+');
    fprintf(fid2,'%11.6f',h1);

    fid3=fopen('C:\temp\resultadosb0.txt','w+');
    fprintf(fid3,'%11.6f',b0);

    fid4=fopen('C:\temp\resultadosb1.txt','w+');
    fprintf(fid4,'%11.6f',b1);


end
```

4.4.2 OBTENCIÓN DE TENSIONES Y DESPLAZAMIENTOS.

Una vez definidas las variables y guardadas en los archivos .txt hacemos correr el Script escrito en lenguaje Python en ABAQUS mediante el siguiente comando. Hacemos referencia a la carpeta donde se encuentra guardado dicho archivo que será en `C:\Temp` y al nombre del Script: `ScriptFinal.py`.

```
%%%%%%%%%% EJECUTAMOS EL SCIPT DE ABAQUS PARA OBTENER %%%%%%%%%%%
%%%%%%%%%% SOLUCION DE DESPLAZAMIENTOS Y TENSIONES    %%%%%%%%%%%

system(['cd C:\Temp & abq692se cae noGUI=ScriptFinal.py']);
```

Cuando se ejecuta el Script escrito en lenguaje Python para ABAQUS se crearan dos archivos de extensión .rpt. Estos dos archivos los crea ABAQUS y reportan información de los nodos de mallado sobre la Tensión de Von Misses (VonMisses.rpt) y los Desplazamientos (U2.rpt) que se producen en ellos. Los

necesitaremos para aplicar las restricciones. El funcionamiento del Script escrito en lenguaje Python para ABAQUS se explica más adelante.

4.4.3 RESTRICCIONES

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%  RESTRINCIONES %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

Pumax=0;
PVMmax=0;
Pulocal=0;
PVMmPuntual=0;
PVMmPuntual2=0;
```

Habrá dos tipos de restricciones:

- Restricciones locales: `Pulocal=0;` `PVMmPuntual=0;` `PVMmPuntual2=0;` que se ampliarán con cada nodo que no cumpla el criterio seleccionado, límite de elástico del material y Deformación máxima permitida. Tendrán una penalización de 1000 kg por cada nodo que no cumpla esta condición.

- Restricciones de Máximas o de Conformidad: `Pumax=0;` `PVMmax=0;` que aplican a la función objetivo una penalización de 10000kg cada uno por la existencia de algún nodo que no cumpla con las condiciones.

4.4.3.1 Restricción por Desplazamiento.

```
%%%%%%%%%% DESPLAZAMIENTO DE LA VIGA %%%%%%%%%%%%%%%%%%%%%%%%%%

%%Restrincion mas fuerte para los valores maximos

hacernada=0;

while fopen('C:\Temp\U2.rpt')==-1
    hacernada=hacernada;
end
```

El siguiente paso será hacer esperar al programa mientras ABAQUS crea los archivos. Para ellos creamos una variable llamada `hacernada=0;` e iniciamos un bucle while que se repetirá hasta que el archivo se halla creado, ya que mientras el programa no se pueda abrir (porque no existe aún) el comando `fopen('C:\Temp\U2.rpt')` nos devolverá el valor -1.

Una vez se ha creado el archivo procedemos a ejecutar su apertura y asígnale una variable.

```
Umax=fopen('C:\Temp\U2.rpt','r+');
```

Para leer el archivo ejecutamos un bucle while que recorrerá el archivo hasta la última línea alojando los datos en una matriz, `C`. Utilizaremos para ello el comando `feof(fid)` que comprueba si el indicador de posición ha llegado al final del archivo, devuelve 1 si el indicador de posición está situado al final del fichero con indicador `fid`, en nuestro caso `Umax` (previamente abierto), y 0 en caso contrario.

La matriz `C` se llenará leyendo el archivo línea a línea utilizando el comando `fgetl(Umax)` e introduciendo el valor de cada línea en una posición de la matriz en formato cell string.

```
n=1;
while ~feof(Umax)
   leer_linea = fgetl(Umax);
   C(n)= cellstr(leer_linea);
   n=n+1;
end
```

A continuación convertimos la matriz C en cadena y posteriormente en número string y hacemos la traspuesta.

```
C=char(C);

C=num2str(C);

C=C';
```

Una vez tenemos la matriz en el formato que deseamos pasamos a realizar una operación compleja que consiste en localizar dentro de la matriz donde se encuentran los valores que corresponden a los desplazamientos. Empezaremos por encontrar los valores máximos y mínimos con los que realizaremos una restricción de conformidad. Para ello buscamos dentro de la matriz donde se encuentra la palabra “Maximum”. El procedimiento será comparar cada letra con la matriz C, transformando la matriz en una matriz de ceros y unos, donde el 1 será que está la letra buscada y el 0 que no coincide. Una vez creadas una matriz para cada letra las modificaremos de tal forma que le añadiremos un cero a medida que avanzamos de letra, de esta forma conseguiremos que al sumar las matrices haya una posición en la que aparezca el número 7, que será el valor máximo que hallaremos en la matriz y corresponderá con la posición en que comienza la palabra “Maximum” dentro de la matriz.

Como conocemos el formato de guardado de ABAQUS sabemos que el valor máximo comienza a partir de 19 espacios desde la letra “M” de la palabra “Maximum” por lo que ya podemos obtener dicho valor.

```
Maximo1=C=='M';
Maximo2=C=='a';
Maximo3=C=='x';
Maximo4=C=='i';
Maximo5=C=='m';
Maximo6=C=='u';
Maximo7=C=='m';

Maximo1=[Maximo1(1:end),0];
Maximo2=[Maximo2(2:end),0,0];
Maximo3=[Maximo3(3:end),0,0,0];
Maximo4=[Maximo4(4:end),0,0,0,0];
Maximo5=[Maximo5(5:end),0,0,0,0,0];
Maximo6=[Maximo6(6:end),0,0,0,0,0,0];
Maximo7=[Maximo7(7:end),0,0,0,0,0,0,0];

MaximoTotal=Maximo1+Maximo2+Maximo3+Maximo4+Maximo5+Maximo6+Maximo7;
PosicionMaximum=find(MaximoTotal==max(MaximoTotal));
PosicionValorMaximo=PosicionMaximum+19;
```

También sabemos que el valor ocupa 10 espacios por lo que le estraemos de la matriz y le transformamos en número decimal.

```
Umaxima=C(PosicionValorMaximo:PosicionValorMaximo+10);
UmaximaFinal=str2num(Umaxima);
```

Realizamos el mismo procedimiento con el valor mínimo del desplazamiento:

```
Minimo1=C=='M';
Minimo2=C=='i';
Minimo3=C=='n';
Minimo4=C=='i';
Minimo5=C=='m';
Minimo6=C=='u';
Minimo7=C=='m';

Minimo1=[Minimo1(1:end),0];
Minimo2=[Minimo2(2:end),0,0];
Minimo3=[Minimo3(3:end),0,0,0];
Minimo4=[Minimo4(4:end),0,0,0,0];
Minimo5=[Minimo5(5:end),0,0,0,0,0];
Minimo6=[Minimo6(6:end),0,0,0,0,0,0];
Minimo7=[Minimo7(7:end),0,0,0,0,0,0,0];

MinimoTotal=Minimo1+Minimo2+Minimo3+Minimo4+Minimo5+Minimo6+Minimo7;
PosicionMinimum=find(MinimoTotal==max(MinimoTotal));
PosicionValorMinimo=PosicionMinimum+19;

Uminima=C(PosicionValorMinimo:PosicionValorMinimo+10);
UminimaFinal=str2num(Uminima);
```

Una vez obtenidos nos quedaremos con el que sea mayor en valor absoluto y aplicamos la restricción en caso de no cumplir con el criterio.

```
if UAbsoluta>0.003 %Yo elijo el valor del desplazamiento máxi-
mo(estaria bien poder introducirlo en la interfaz grafica)
    Pumax=10000;
end
```

A continuación se leerán el desplazamiento de cada nodo utilizando el mismo método que para los valores máximos. Se buscará la palabra “@Loc” que sólo aparece una vez en el archivo exportado y conociendo el formato se buscaran los valores recorriendo la matriz C. En este caso la penalización será de 1000kg por cada nodo que no cumpla con el criterio.

```
%%Restricciones mas moderada para los limites localizados en los nodos

Maximo1local=C=='@';
Maximo2local=C=='L';
Maximo3local=C=='o';
Maximo4local=C=='c';


Maximo1local=[Maximo1local(1:end),0];
Maximo2local=[Maximo2local(2:end),0,0];
Maximo3local=[Maximo3local(3:end),0,0,0];
Maximo4local=[Maximo4local(4:end),0,0,0,0];


MaximoLocal=Maximo1local+Maximo2local+Maximo3local+Maximo4local;
PosicionMaximumLocal=find(MaximoLocal==max(MaximoLocal));
PosicionValorMaximoLocal=PosicionMaximumLocal+150;


while (PosicionMinimum-227)>=PosicionValorMaximoLocal
UmaximaLocal=C(PosicionValorMaximoLocal:PosicionValorMaximoLocal+16);
PosicionValorMaximoLocal=PosicionValorMaximoLocal+16;

UmaximaFinalLocal=str2num(UmaximaLocal);

    if abs(UmaximaFinalLocal)>0.003
        Pulocal=Pulocal+1000;
    end

PosicionValorMaximoLocal=PosicionValorMaximoLocal+64;
UmaximaLocal=C(PosicionValorMaximoLocal:PosicionValorMaximoLocal+16);

end
```

Finalmente se cierra y elimina el archivo, ya que en caso de no ser así al ejecutar el Script de Python con el siguiente individuo se reportaría el archivo sin sustituir al anterior quedando los resultados de todos los individuos guardados y no nos valdría el método empleado para la obtención de resultados.

```
fclose('all');

delete('C:\Temp\U2.rpt');
```

4.4.3.2 Restricción por Plastificación del Material.

Para saber si el material de la viga plastifica aplicaremos el criterio de Von Mises que establece que "la plastificación se produce cuando el segundo invariante del tensor de tensiones desviadoras es igual al cuadrado de una constante k' propiedad del material". Es decir, si la tensión equivalente de Von Misses supera el límite elástico del material éste plastifica.

El valor de la tensión le obtendremos del archivo VonMisses.rpt utilizando el mismo método que para el desplazamiento de la viga. En primer lugar el programa permanecerá en espera hasta detectar que el archivo ya se ha creado.

```
%%%%%%%%%% PLASTIFICACIÓN DEL MATERIAL %%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%  sigman<sigmay    y    tau<(sigmay/sqrt(3))    %%%%%%%%%%%%%%%%%%
%%% Abaqus nos proporciona la energia de von misses por lo tanto %%%
%%% el criterio de plastificación será:                          %%%
%%% Si sigmaVM>sigmay el material plastifica                     %%%

hacernada=0;

while fopen('C:\Temp\VonMises.rpt')==-1
    hacernada=hacernada;
end
```

A continuación abrimos el archivo y lo leemos guardando el contenido en una matriz, en este caso la matriz será `C2`. Realizamos el mismo procedimiento para determinar dónde se encuentran los valores máximos y mínimos buscando la posición de las palabras. El problema que nos encontramos ahora con respecto al anterior archivo es que el formato de guardado de ABAQUS es distinto, habrá un valor máximo y mínimo para cada región, en nuestro caso habrá un máximo y un mínimo para cada sección de la viga.

```
n=1;
while ~feof(Mises)
   leer_linea = fgetl(Mises);
   C2(n)= cellstr(leer_linea);
   n=n+1;
end

C2=char(C2);

C2=num2str(C2);
C2=C2';

Maximo1=C2=='M';
Maximo2=C2=='a';
Maximo3=C2=='x';
Maximo4=C2=='i';
Maximo5=C2=='m';
Maximo6=C2=='u';
Maximo7=C2=='m';

Maximo1=[Maximo1(1:end),0];
Maximo2=[Maximo2(2:end),0,0];
Maximo3=[Maximo3(3:end),0,0,0];
Maximo4=[Maximo4(4:end),0,0,0,0];
Maximo5=[Maximo5(5:end),0,0,0,0,0];
Maximo6=[Maximo6(6:end),0,0,0,0,0,0];
Maximo7=[Maximo7(7:end),0,0,0,0,0,0,0];

MaximoTotal=Maximo1+Maximo2+Maximo3+Maximo4+Maximo5+Maximo6+Maximo7;
PosicionMaximum=find(MaximoTotal==max(MaximoTotal));
PosicionValorMaximo=PosicionMaximum+19;
```

Por esta razón la variable `PosicionValorMaximo` no almacena un único valor, será un vector que almacenará todas las posiciones en las que se encuentra la palabra “Maximum”. Por lo tanto habrá que crear un vector que almacene todos los valores de las tensiones máximas con un bucle for, a continuación se comprueba si alguno de estos valores supera el límite elástico y de ser así se le aplica la penalización.

```
VMmaximaFinal=1:1:NSecciones;


for i=1:NSecciones
    VMmaximaFinal(i)=str2num(C2(PosicionValorMaximo(i):
                                PosicionValorMaximo(i)+10));
end

LVMmf2=length(VMmaximaFinal);

for i=1:LVMmf2;
    if abs(VMmaximaFinal(i))>LimiteElastico
        PVMmPuntual2=PVMmPuntual2+1000;
    end
end
```

Realizamos el mismo procedimiento para los valores mínimos y tendremos aplicadas las penalizaciones locales del criterio de plastificación.

```
VMmaximaFinal=max(VMmaximaFinal);
VMmaximaFinal=abs(VMmaximaFinal);

Minimo1=C2=='M';
Minimo2=C2=='i';
Minimo3=C2=='n';
Minimo4=C2=='i';
Minimo5=C2=='m';
Minimo6=C2=='u';
Minimo7=C2=='m';

Minimo1=[Minimo1(1:end),0];
Minimo2=[Minimo2(2:end),0,0];
Minimo3=[Minimo3(3:end),0,0,0];
Minimo4=[Minimo4(4:end),0,0,0,0];
Minimo5=[Minimo5(5:end),0,0,0,0,0];
Minimo6=[Minimo6(6:end),0,0,0,0,0,0];
Minimo7=[Minimo7(7:end),0,0,0,0,0,0,0];

MinimoTotal=Minimo1+Minimo2+Minimo3+Minimo4+Minimo5+Minimo6+Minimo7;
PosicionMinimum=find(MinimoTotal==max(MinimoTotal));
PosicionValorMinimo=PosicionMinimum+19;

VMminimaFinal=1:1:NSecciones;

for i=1:NSecciones
    VMminimaFi-
nal(i)=str2num(C2(PosicionValorMinimo(i):PosicionValorMinimo(i)+10));
end

LVMmf=length(VMminimaFinal);

for i=1:LVMmf
    if abs(VMminimaFinal(i))>LimiteElastico
        PVMmPuntual=PVMmPuntual+1000;
    end
end
```

A continuación aplicamos la penalización de conformidad de mayor valor en caso de que no haya ninguna sección que supere el límite elástico, cerramos el archivo y le eliminamos para dejar sitio al próximo individuo.

```
VMminimaFinal=max(VMminimaFinal);
VMminimaFinal=abs(VMminimaFinal);


VMAbsoluta=max(VMminimaFinal,VMmaximaFinal)
```

4.4.4 CÁLCULO DEL PESO DE LA VIGA

Para terminar con el Script se calcula el peso final de la viga y se le suman las penalizaciones. El cálculo del peso se realiza calculando el Área de cada una de las secciones, posteriormente el volumen sabiendo la longitud de cada sección y finalmente el peso de la viga con la densidad.

```
%Calculamos el Peso de la Viga

if strcmp(Seccion,'Tubular')==1
    Area=pi*((rs).^2-(rs-ts).^2);
end

if strcmp(Seccion,'Rectangular')==1
    for i=1:NSecciones
        Area(i)=ar(i)*br(i);
    end
end

if strcmp(Seccion,'Caja')==1
    for i=1:NSecciones
        Area(i)=ac(i)*bc(i)-((ac(i)-tc(i)*2)*(bc(i)-tc(i)*2));
    end
end

if strcmp(Seccion,'DobleT')==1
    for i=1:NSecciones
        Area(i)=h0(i)*b0(i)+2*h1(i)*b1(i);
    end
end



Volumen=Area*LongitudSeccion;
Peso=Volumen*Densidad;
PesoTotal=0;

for i=1:NSecciones
    PesoTotal=PesoTotal+Peso(i);
end


%Funcion Objetivo: Peso de la viga
y=PesoTotal+Pumax+Pulocal+PVMmax+PVMmPuntual+PVMmPuntual2
```

4.5 SCRIPT EN LENGUAJE PYTHON PARA ABAQUS

Antes de proceder a la explicación del script cabe destacar que la escritura de los cuadros de texto carece de acentuación, esto es debido a que el lenguaje de programación de Python tiene problemas con determinados caracteres, entre ellos los acentos.

La función de este Script es dar a ABAQUS las instrucciones para construir la viga que hemos diseñado en nuestra interfaz gráfica, hacer un análisis estructural y reportarnos información sobre su comportamiento. En este apartado están detallados tanto el funcionamiento del programa como usuario como el funcionamiento del Script.

El archivo creado será un archivo con extensión ".py" que es el lenguaje de programación con el que trabaja ABAQUS. Para poder interactuar Python con Excel es necesario instalar en la Computadora los módulos necesarios. Para instalar los módulos hay que seguir las siguientes instrucciones de lo contrario tendremos problemas a la hora de ejecutar el Script con ABAQUS.

Para la version de ABAQUS utilizada (6.9-2SE) en el proyecto requiere la instalación del Python 2.7.8 con las siguientes versiones de módulos (los módulos se deben instalar obligatoriamente pero si no son exactamente estas versiones puede desembocar en errores a la hora de ejecutar el script):

-xlrd-0.7.1.

-xlwt-0.7.5

-xlutils-1.4.1.

Cuando el problema a resolver es simplemente una pieza o una estructura determinada suele ser mucho más práctico e intuitivo crear la pieza utilizando ABAQUS y las barra de herramientas que posee el programa. En nuestro caso, como la estructura varía constantemente en cada individuo de los AGs es mucho más práctico hacer un Script que nos construya y simule el comportamiento de la estructura sin necesidad siquiera de abrir el módulo ABAQUS/CAE.

A continuación se procede a explicar el funcionamiento del Script acompañado de cómo se realizaría utilizando manualmente el programa para hacerlo más didáctico:

4.5.1 CREACCIÓN, LECTURA Y DEFINICIÓN DE VARIABLES.

```
#*********************************************************************
*****************************
# Script Viga Final
#*********************************************************************
*****************************

import math
import xlrd
import xlwt
import xlutils
```

`import math e import xlutils`: Importa el módulo de comandos matemáticos y comandos prácticos.

`import xlwt e import xlrd`: Importa los módulos que permiten escribir y leer en Microsoft Excel respectivamente.

```
###############   LECTURA DE VARIABLES DESDE EXCEL      ###############

Lectura = xlrd.open_workbook("ExcelProyecto.xls")
HojaParametros = Lectura.sheet_by_index(0)
```

El comando `xlrd.open_workbook` abre el libro de Excel.

`"ExcelProyecto.xls"` es el archivo de Excel donde se encuentran guardados los parámetros de las variables que necesitamos para construir la viga.

`Lectura.sheet_by_index(0)` abre la Hoja de Excel.

```
############       DEFINICION DE VARIABLES       ###############

Longitud=HojaParametros.cell_value(rowx=4, colx=2)

NSecciones=HojaParametros.cell_value(rowx=5, colx=2)

LongitudSeccion=Longitud/float(NSecciones)

LongitudSeccion2=Longitud/float(NSecciones)

Densidad=HojaParametros.cell_value(rowx=6, colx=2)

ModuloYoung=HojaParametros.cell_value(rowx=7, colx=2)

CoeficientePoison=HojaParametros.cell_value(rowx=8, colx=2)

Contador0=0

Contador1=1

Vacio=()

############## LOS TIPOS DE PERFIL PUEDEN SER:  ##############
#########'DobleT', 'Rectangular', 'Tubular' o 'Caja' ########

TipoPerfil=HojaParametros.cell_value(rowx=9, colx=2)
```

`HojaExecel.cell_value(rowx=numero fila, colx=numero columna)`: Este comando lee el valor de la celda indicando el nombre de la variable que ha guardado la lectura de la Hoja en la que se encuentra la celda, e indicando la fila y la columna. Para Python tanto las filas como las columnas se enumeran partiendo de 0 y ambas de forma numérica, al contrario que ocurre en MATLAB.

`Longitud`: Es la longitud total de la viga.

`NSecciones`: Llamado también `Grado de discretización`, es el número de partes en que se divide la viga y a la que se le asigna una sección distinta.

`LongitudSeccion`: Es la longitud de cada una de las secciones en que se divide la viga, será la misma para todas ya que se divide en partes iguales.

`LongitudSeccion2`: Tiene el mismo valor que `Longitud de Seccion` pero nos interesa tener ambos debido en que a lo largo del script se modifica la variable.

`Densidad`: Es la densidad del material de la viga.

`ModuloYoung`: Es el Módulo de Young del material de la viga.

`CoeficientePoison`: Es el Coeficinete de Poisson del material de la viga.

`Contador1, Contador2 y Vacio()`: Son variables que se utilizarán con fines de programación que no afectan a la estructura de la viga.

`TipoPerfil`: Variable que asume el valor del tipo de perfil de la viga.

```
##################      Tipo de perfil DobleT         ##################

if TipoPerfil=='DobleT':
      separacion=11
      fo = open("C:/temp/resultadosh0.txt", "r+")
      str1 = fo.read()

      fo = open("C:/temp/resultadosh1.txt", "r+")
      str2 = fo.read()

      fo = open("C:/temp/resultadosb0.txt", "r+")
      str3 = fo.read()

      fo = open("C:/temp/resultadosb1.txt", "r+")
      str4 = fo.read()

      h1s = []
      h2s = []
      b1s = []
      b2s = []


      while Contador1<=NSecciones:
            h1s.append(Vacio)

      h1s[Contador0]=float(str1[3+Contador0*separacion:separacion*Contador1])

            h2s.append(Vacio)

      h2s[Contador0]=float(str2[3+Contador0*separacion:separacion*Contador1])

            b1s.append(Vacio)

      b1s[Contador0]=float(str3[3+Contador0*separacion:separacion*Contador1])

            b2s.append(Vacio)

      b2s[Contador0]=float(str4[3+Contador0*separacion:separacion*Contador1])

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
Contador1=1
```

```
############       Tipo de perfil Tubular        #############

if TipoPerfil=='Tubular':
      separacion=11
      fo = open("C:/temp/resultadosrs.txt", "r+")
      str1 = fo.read()

      fo = open("C:/temp/resultadosts.txt", "r+")
      str2 = fo.read()

      rs = []
      ts = []


      while Contador1<=NSecciones:
            rs.append(Vacio)

      rs[Contador0]=float(str1[3+Contador0*separacion:separacion*Contador1])

            ts.append(Vacio)

      ts[Contador0]=float(str2[3+Contador0*separacion:separacion*Contador1])

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
Contador1=1
```

```
#############          Tipo de perfil Rectangular          ##############

if TipoPerfil=='Rectangular':
      separacion=11
      fo = open("C:/temp/resultadosar.txt", "r+")
      str1 = fo.read()

      fo = open("C:/temp/resultadosbr.txt", "r+")
      str2 = fo.read()

      s = []
      bs = []


      while Contador1<=NSecciones:
            s.append(Vacio)

      s[Contador0]=float(str1[3+Contador0*separacion:separacion*Contador1])

            bs.append(Vacio)

      bs[Contador0]=float(str2[3+Contador0*separacion:separacion*Contador1])

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
Contador1=1
```

```
#################       Tipo de perfil Caja          ##################


if TipoPerfil=='Caja':
      separacion=11
      fo = open("C:/temp/resultadosac.txt", "r+")
      str1 = fo.read()

      fo = open("C:/temp/resultadosbc.txt", "r+")
      str2 = fo.read()

      fo = open("C:/temp/resultadostc.txt", "r+")
      str3 = fo.read()

      acs = []
      bcs = []
      tcs = []


      while Contador1<=NSecciones:
            acs.append(Vacio)

      acs[Contador0]=float(str1[3+Contador0*separacion:separacion*Contador1])

            bcs.append(Vacio)

      bcs[Contador0]=float(str2[3+Contador0*separacion:separacion*Contador1])

            tcs.append(Vacio)

      tcs[Contador0]=float(str3[3+Contador0*separacion:separacion*Contador1])

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
Contador1=1
```

En esta parte del Script se definen las variables que contendrán los valores de las secciones de cada perfil. Los valores de los perfiles están guardados en un archivo .txt, procedentes de MATLAB creados y optimizados por el Script de Algoritmos Genéticos.

Comenzamos con una sentencia IF para identificar el tipo de sección.

Los valores están guardados en el archivo de forma organizada para facilitar su lectura, que se encuentra descrita en el apartado del script de AGs. Los valores quedan guardados de forma que ocupan 11 posiciones de las cuales 6 posiciones se destinan a la parte decimal por eso definimos la variable `separacion=11`.

Por ejemplo en una viga de 7 secciones los datos quedarían guardados de la siguiente forma:

0.684897 0.354252 0.723754 0.186364 0.560264 0.670235 0.272141

Los siguiente comandos `open()` y `.read()` se utilizan para abrir y leer respectivamente el archivo .txt. Es importante dotar al comando `open()` de `"r+"` ya que nos permite la lectura y escritura del archivo y sitúa el puntero al principio. La lectura de los archivos quedará guardada en las variables `str1 str2 str3 str4` en formato string que se pasaran luego a flotante.

Posteriormente están definidas las variables de cada parámetro(alto, ancho, radio, espesor…) de la sección. Cada variable corresponde a un vector vacío donde se almacenarán los datos de cada parámetro.

Los parámetros que provienen de los AGs no se corresponden con los parámetros que pide insertar ABAQUS por lo que se realizan algunas modificaciones:

PERFIL DOBLE T:

```
h1s = h-t1-t2
h2s = t1 = t2
b1s = t3
b2s = b1 = b2
```

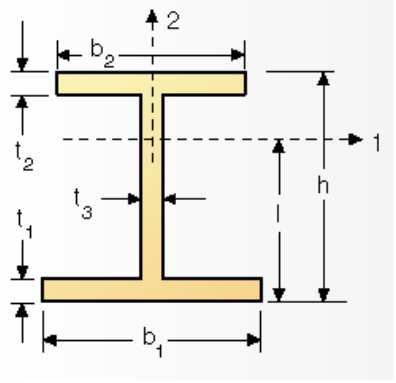


Figura 4.5.1.1: Párametros a insertar en ABAQUS para el perfil Doble T.

PERFIL TUBULAR:

```
rs = r
ts = t
```

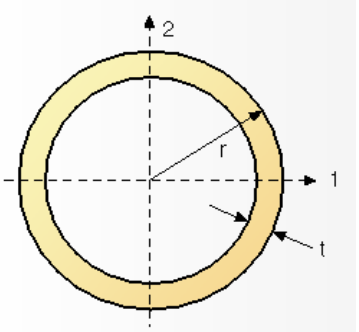


Figura 4.5.1.2: Párametros a insertar en ABAQUS para el perfil Tubular.

PERFIL RECTANGULAR:

```
s = a
bs = b
```

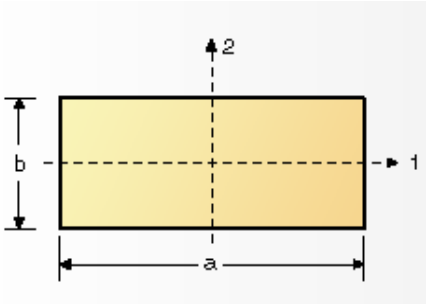


Figura 4.5.1.3: Párametros a insertar en ABAQUS para el perfil Rectangular.

PERFIL CAJA:

```
acs = a
bcs = b
tcs = t1 = t2
```

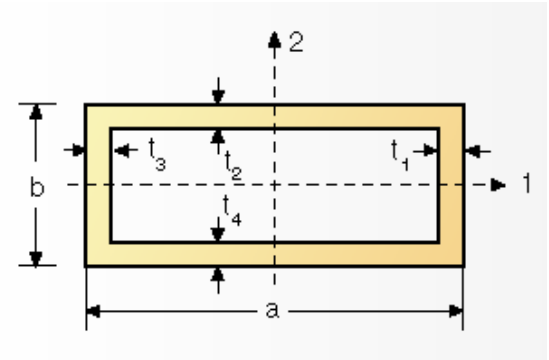


Figura 4.5.1.4: Párametros a insertar en ABAQUS para el perfil Rectangular.

A continuación se ejecuta una sentencia WHILE que se ejecutara NSecciones veces. Primero tenemos el comando `.append()` que se aplica a cada una de las variables de las longitudes con la variable `Vacio`, lo que hacemos con esto es añadirle una posición sin nada a cada vector. Por ejemplo para la variable `bcs` que es el ancho del perfil Caja hacemos: `bcs.append(Vacio)`.

Ahora para guardar los valores en el vector hacemos, por ejemplo, para la variable `acs`.

```
    acs[Contador0]=float(str1[3+Contador0*separacion:separacion*Contad
or1])
```

El Contador0 tiene un valor inicial de 0 e irá aumentando su valor de uno en uno para referirnos a todas las posiciones del vector empezando por la primera y le almacenamos el valor en decimal obtenido del archivo .txt.

```
############       NOMBRES DE PERFIL Y SECCION   ###############

NamesProfiles=[]
NamesSecciones=[]

while Contador0<=NSecciones:
      NamesProfiles.append(Vacio)
      NamesProfiles[Contador0] = 'Profile'+str(Contador1)
      NamesSecciones.append(Vacio)
      NamesSecciones[Contador0] = 'Seccion'+str(Contador1)
      Contador0 = Contador0 + 1
      Contador1=Contador1+1

Contador0=0
Contador1=1
```

En ABAQUS para poder asignar a cada tramo una sección se necesita haberla creado y definirla con un nombre, y para poder asignar un perfil a una sección ídem. Para ello crearemos de la misma forma que para los parámetros de las secciones un vectores en los que en cada posición aparezca el nombre de cada una de las secciones, habiendo tantos nombres como secciones y tantos nombres como perfiles.

La diferencia con respecto a las variables de los parámetros es que ahora los valores tienen formato de cadena de caracteres, por lo que necesitamos utilizar el comando `str()`, para cambiar el formato de un número a cadena de caracteres y así poder crear cienes de nombres fácilmente.

```
#############          CARGAS SOBRE LA VIGA          #################

NCargaPuntual=HojaParametros.cell_value(rowx=4, colx=27)

NCargaLineal=HojaParametros.cell_value(rowx=5, colx=27)

NMomentoPuntual=HojaParametros.cell_value(rowx=6, colx=27)
```

Pasamos a definir las cargas, y en primer lugar leemos desde Excel el número total de cargas existentes de cada tipo.

`NCargaPuntual`: Número de cargas puntuales

`NCargaLineal`: Número de cargas lineales

`NMomentoPuntual`: Número de momentos puntuales.

```
################        Carga Puntual           #############

CargaPuntual=[]

PosicionCargaPuntual=[]

NamesCargaPuntual=[]

while Contador0<NCargaPuntual:
      CargaPuntual.append(Vacio)
      CargaPuntual[Contador0]=HojaParametros.cell_value(rowx=4+Contador0,
      colx=30)
      PosicionCargaPuntual.append(Vacio)
      PosicionCargaPuntual[Contador0]=HojaParametros.cell_value(rowx=4+Cont
      ador0, colx=31)
      NamesCargaPuntual.append(Vacio)
      NamesCargaPuntual[Contador0]='CargaPuntual'+str(Contador0)
      Contador0=Contador0+1

Contador0=0
```

```
############            Carga Lineal o Distribuida          ###############

CargaLineal=[]

PosicionInicialCargaLineal=[]

PosicionFinalCargaLineal=[]

NamesCargaLineal=[]

while Contador0<NCargaLineal:
      CargaLineal.append(Vacio)
      CargaLineal[Contador0]=HojaParametros.cell_value(rowx=4+Contador0,
      colx=32)
      PosicionInicialCargaLineal.append(Vacio)
      PosicionInicialCargaLineal[Contador0]=HojaParametros.cell_value(rowx=
      4+Contador0, colx=33)
      PosicionFinalCargaLineal.append(Vacio)
      PosicionFinalCargaLineal[Contador0]=HojaParametros.cell_value(rowx=4+
      Contador0, colx=34)
      NamesCargaLineal.append(Vacio)
      NamesCargaLineal[Contador0]='CargaLineal'+str(Contador0)
      Contador0=Contador0+1

Contador0=0
```

```
#######################       Momento Puntual         ###################

MomentoPuntual=[]

PosicionMomentoPuntual=[]

NamesMomentoPuntual=[]

while Contador0<NMomentoPuntual:
      MomentoPuntual.append(Vacio)
      MomentoPuntual[Contador0]=HojaParametros.cell_value(rowx=4+Contador0,
      colx=35)
      PosicionMomentoPuntual.append(Vacio)
      PosicionMomentoPuntual[Contador0]=HojaParametros.cell_value(rowx=4+
      Contador0, colx=36)
      NamesMomentoPuntual.append(Vacio)
      NamesMomentoPuntual[Contador0]='MomentoPuntual'+str(Contador0)
      Contador0=Contador0+1

                                    Contador0=0
```

En esta parte del script se realiza el mismo procedimiento que con los parámetros de los perfiles, se leen y almacenan los valores de las cargas, su posición respecto a la viga y se crea un nombre para cada una, pero con la gran diferencia de que los datos se leen desde Excel facilitando mucho el procedimiento.

```
################         APOYOS Y EMPOTRAMIENTOS             #################

NApoyosSimples=HojaParametros.cell_value(rowx=7, colx=27)

NEncastres=HojaParametros.cell_value(rowx=8, colx=27)


##################            Apoyos            ##########################

PosicionApoyoSimple=[]

NamesApoyoSimple=[]

while Contador0<NApoyosSimples:
      PosicionApoyoSimple.append(Vacio)
      PosicionApoyoSimple[Contador0]=HojaParametros.cell_value(rowx=4+
      Contador0, colx=38)
      NamesApoyoSimple.append(Vacio)
      NamesApoyoSimple[Contador0]='ApoyoSimple'+str(Contador0)
      Contador0=Contador0+1

Contador0=0


################        EMPOTRAMIENTOS          ############################


PosicionEncastre=[]

NamesEncastre=[]

while Contador0<NEncastres:
      PosicionEncastre.append(Vacio)
      PosicionEncastre[Contador0]=HojaParametros.cell_value(rowx=4+Contador0,
       colx=39)
      NamesEncastre.append(Vacio)
      NamesEncastre[Contador0]='Encastre'+str(Contador0)
      Contador0=Contador0+1

Contador0=0
```

Idéntico procedimiento que para las cargas se realiza en este tramo para las coacciones de la viga, apoyos y empotramientos.

4.5.2 RESOLUCIÓN DEL PROBLEMA EN ABAQUS.

```
#### Una vez definidas las variables comenzamos a realizar la viga ####

from abaqus import*
from abaqusConstants import*
import regionToolset


session.viewports['Viewport: 1'].setValues(displayedObject=None)
```

Ya definidas las variables comenzamos a trabajar con ABAQUS, comenzando con importar los módulos principales e iniciando la sesión de inicio con el comando que podemos apreciar en el cuadro superior.

A continuación, y con la ayuda del Manual de ABAQUS del departamento de Mecánica de Medios Continuos y Teoría de Estructuras se procede a explicar el Script:

Para resolver el problema en ABAQUS se utilizan nueve módulos donde se definen las características del problema. De momento sólo hemos declarado las variables que tendremos que utilizar para que se ejecute el programa.

```
#-------------------------------------------------------------

#Creaccion del Modelo
mdb.models.changeKey(fromName='Model-1', toName='Viga')
beamModel = mdb.models['Viga']
```

Se crea el modelo con el nombre de “Viga”.

4.5.2.1 Módulo PART.

El módulo “Part” hace referencia a las partes que componen el problema, en nuestro caso se trata de una sola parte. Nuestro problema consiste en una viga continua, lineal y elástica por lo que para representarla utilizaremos diseño en el plano (2D) y asignaremos que se va a crear un cuerpo deformable. En el cuadro siguiente se ven programadas estas características.

```
#-------------------------------------------------------------
#Create the Part

import sketch
import part

#Definimos el tipo de viga que deseamos

beamPart = beamModel.Part(dimensionality=TWO_D_PLANAR, name='Beam',
type=DEFORMABLE_BODY)
```

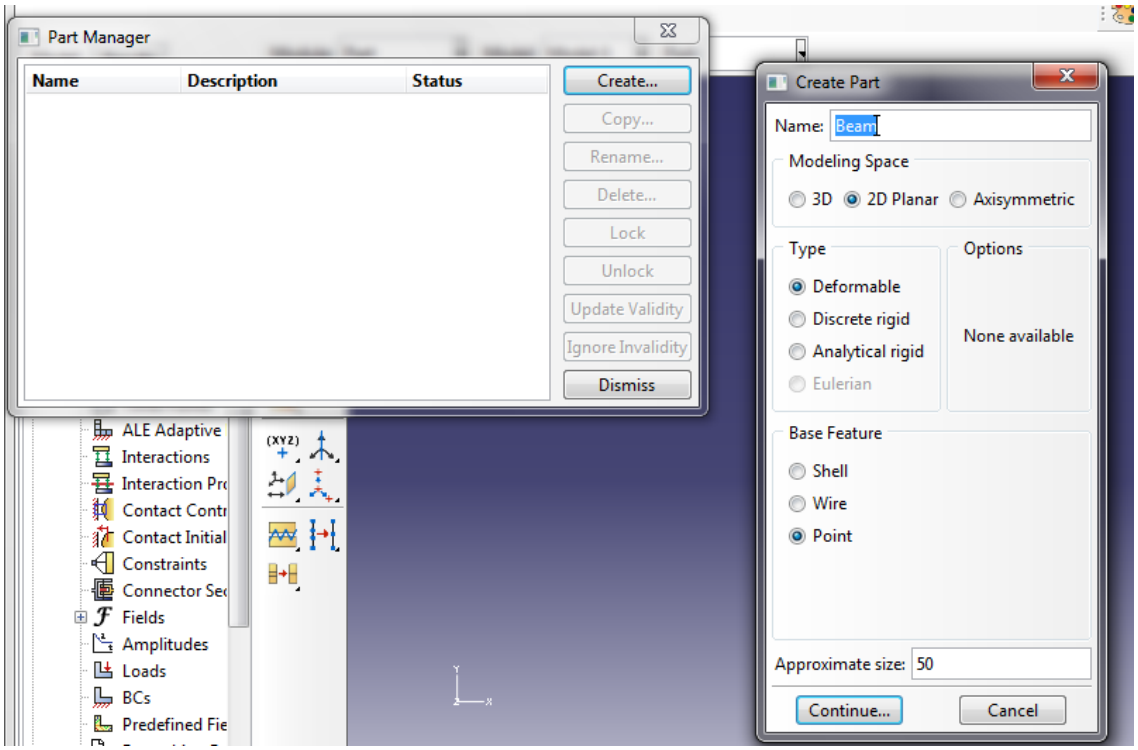


Figura 4.5.2.1.1: Cuadros de diágolo en el módulo PART para crear y editar una "part".

Para continuar con la generación se crean tantos Datum Point como número de secciones más una haya. Podemos acceder a la opción de datum point desde los botones de accesos rápidos del módulo Part. Mientras que en el Script se programa la posición de los puntos con una sentencia "while" y utilizando las Variables: `LongitudSeccion`, `LongitudSeccion2` y `Longitud`. En la posición (0,0,0) ya tenemos el punto de referencia y se colocan los siguientes puntos a una distancia de `LongitudSeccion`, que recordamos que es la longitud total de la viga entre en número de secciones.

```
#Creamos la viga con puntos dependiendo del numero de secciones y la longitud
de dicha viga.
#Para ello utilizaremos la LongitudSeccion y un contador con el NSecciones.

while LongitudSeccion<Longitud:
    LongitudSeccion=LongitudSeccion2*Contador0
    beamPart.DatumPointByCoordinate(coords=(LongitudSeccion,0,0))
    Contador0=Contador0+1

#Volvemos a colocar los valores iniciales a las variabless

Contador0=0
Contador1=1
LongitudSeccion=Longitud/float(NSecciones)
```

Una vez establecidos los puntos hay que unirlos con “hilos” (WIRE). Se elige WIRE porque se trata de una estructura formada por barras unidimensionales, si se tratase de un sólido tridimensional habría que elegir SOLID. El comando que se utiliza en ABAQUS CAE es Create Wire: Point to Point, que une dos puntos que se marquen.

En el código escrito quedará de la siguiente manera, utilizaremos una sentencia while para recorrer todos los puntos ayudándonos de los Contadores para hacer referencia correctamente a cada punto.

```
#Pasamos a unir los puntos creados para formar la viga
##Comando .sort() se utiliza para ordenar listas

beamPart_datums_keys = beamPart.datums.keys()
beamPart_datums_keys.sort()
datum_points = beamPart.datums


while LongitudSeccion<Longitud:
      beamPart.WirePolyLine(points=((datum_points[beamPart_datums_keys
      [Contador0]],datum_points[beamPart_datums_keys[Contador1]])),
      mergeWire = OFF, meshable = ON)
      Contador1=Contador1+1
      Contador0=Contador0+1
      LongitudSeccion=LongitudSeccion2*Contador0

LongitudSeccion=Longitud/float(NSecciones)
Contador1=1
Contador0=0
```

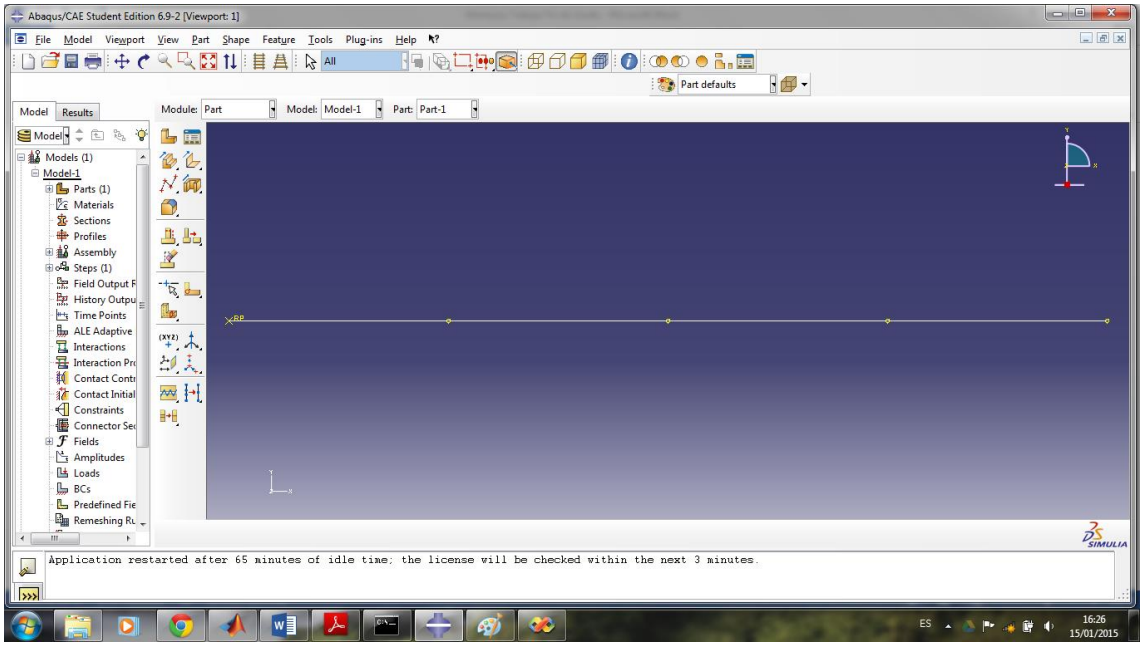


Figura 4.5.2.1.2: “Part” de la estructura

4.5.2.2 Módulo PROPERTY

Ya tenemos creada la geometría de nuestra estructura, en el módulo "PROPERTY" se crea el material y se asigna éste a todos los elementos. El material se crea con las variables anteriormente definidas y leídas de la hoja Excel (Densidad, Módulo de Young y Coeficiente de Poison). En primer hay que poner un nombre al material, el nombre que pondremos será `'Acero s235'` aunque no necesariamente el material deba tener las cualidades de ese acero, es solamente un nombre simbólico. Con lo que se crea un material con propiedades elásticas. Los comandos que utilizamos en el Script para crear el material son los siguientes:

```
#------------------------------------------------------------------------
-------------------------
#Create material

import material

beamMaterial = beamModel.Material(name = 'Acero s235')
beamMaterial.Density(table=((Densidad,),))
beamMaterial.Elastic(table=((ModuloYoung,CoeficientePoison),))
```

Una vez creado el material no podemos asignarlo directamente a la geometría creada. ABAQUS distingue dos conceptos distintos, sección y perfil. El perfil designa a la geometría de la sección, ya hemos definido anteriormente los diferentes tipos de perfil, tan sólo nos queda crearlos con los comandos correspondientes y haciendo referencia a las variables. La sección designa por un lado la clase de estructura, en nuestro caso una viga por lo que será una sección de clase "Beam", y por otro lado el tipo que en este caso será de nuevo tipo "Beam" ya que se trata de una estructura reticulada formada por nudos rígidos que trabajarán a esfuerzo axil, cortante y momento flector. Se puede ver en el código que se muestra a continuación diferenciadas las dos partes, la creación de un perfil con donde se asignan los parámetros con las variables anteriormente definidas y la creación de una Sección que abarca tanto el material como el perfil.

```
 #-------------------------------------------------------------------------
------------------------
#Create a section and asing the beam to it import section

import section

if TipoPerfil=='DobleT':
      while Contador1<=NSecciones:

            beamModel.IProfile(b1=b2s[Contador0],b2=b2s[Contador0],
            h=(h1s[Contador0]+h2s[Contador0]*2),
            l=(h1s[Contador0]+h2s[Contador0]*2)/float(2),
            name=NamesProfiles[Contador0], t1=h2s[Contador0],
            t2=h2s[Contador0], t3=b1s[Contador0])

            beamModel.BeamSection(integration=DURING_ANALYSIS, material='Acero
            s235', name=NamesSecciones[Contador0], poissonRatio=0.0,
            profile=NamesProfiles[Contador0], temperatureVar=LINEAR)

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
Contador1=1
```

```
if TipoPerfil=='Tubular':
      while Contador1<=NSecciones:

            beamModel.PipeProfile(name=NamesProfiles[Contador0],
            r=rs[Contador0], t=ts[Contador0])

            beamModel.BeamSection(integration=DURING_ANALYSIS, material=
            'Acero s235', name=NamesSecciones[Contador0], poissonRatio=0.0,
            profile=NamesProfiles[Contador0], temperatureVar=LINEAR)

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
Contador1=1
```

```
if TipoPerfil=='Rectangular':
      while Contador1<=NSecciones:

            beamModel.RectangularProfile(name=NamesProfiles[Contador0],
            a=s[Contador0], b=bs[Contador0])

            beamModel.BeamSection(integration=DURING_ANALYSIS, material='Acero
            s235', name=NamesSecciones[Contador0], poissonRatio=0.0,
            profile=NamesProfiles[Contador0], temperatureVar=LINEAR)

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
Contador1=1
```

```
if TipoPerfil=='Caja':
      while Contador1<=NSecciones:

            beamModel.BoxProfile(a=acs[Contador0], b=bcs[Contador0],
            name=NamesProfiles[Contador0], t1=tcs[Contador0],
            uniformThickness=ON)

            beamModel.BeamSection(integration=DURING_ANALYSIS, material=
            'Acero s235', name=NamesSecciones[Contador0], poissonRatio=0.0,
            profile=NamesProfiles[Contador0], temperatureVar=LINEAR)

            Contador0=Contador0+1
            Contador1=Contador1+1

Contador0=0
```

Posteriormente tenemos que asignar la sección a la geometría de viga creada. Puesto que hay numerosas secciones tenemos que localizar las regiones donde aplicar cada una de ellas utilizando la variable `PuntoMedioSeccion`.

```
#Asignar una seccion a cada tramo de la viga tipo beam y la orientacion.

PuntoMedioSeccion=LongitudSeccion/float(2)

while Contador1<=NSecciones:

      edges_for_beam_section_assignment1 = beamPart.edges.findAt(((
      PuntoMedioSeccion,0,0),),)

      beam_region1 = regionToolset.Region(edges = edges_for_
      beam_section_assignment1)

      beamPart.SectionAssignment(region=beam_region1,sectionName=
      NamesSecciones[Contador0])

      beamPart.assignBeamSectionOrientation(region = beam_region1,
      method= N1_COSINES, n1=(0,0,-1))

      Contador1=Contador1+1
      Contador0=Contador0+1
      LongitudSeccion=Longitud/float(NSecciones)
      PuntoMedioSeccion=LongitudSeccion/float(2)
      PuntoMedioSeccion=LongitudSeccion*Contador1


LongitudSeccion=Longitud/float(NSecciones)
Contador0=0
Contador1=1
PuntoMedioSeccion=LongitudSeccion/float(2)
```

Para el código se ejecuta una sentencia *while* que se repite hasta NSecciones veces. Las dos primeras líneas dentro del bucle seleccionan un punto de un tramo de viga, más concretamente seleccionan el punto medio de cada sección (valdría cualquier punto del tramo excepto los extremos).

Las siguientes dos líneas crean una región, `beam_region1`, a partir del tramo seleccionado anteriormente, `edges_for_beam_section_assignment1`.

A continuación a esa región se le asigna una sección con su correspondiente perfil y por último se le asigna una dirección para que quede colocado el perfil en la posición deseada.

4.5.2.3 Módulo ASEMBLY

Como su propio nombre indica el módulo “ASEMBLY” se utiliza para unir las diferentes partes que se han creado en el módulo “PART” y convertimos una entidad geométrica en una entidad física (instance). Es una función muy útil ya que si tenemos una estructura con muchas piezas iguales tan sólo habría que crear una única Part y en este módulo crear varias entidades físicas a partir de esa única Part. Al crear una instance debemos elegir entre dependiente o independiente que hace referencia sobre donde se aplicará el mallado, si en la part o en instance. En nuestro caso seleccionamos depend = ON, pero puesto que tenemos una única part nos sería indiferente.

```
#-------------------------------------------------------------------
#Create the Assembly

import assembly

#Create the part instance

beamAssembly = beamModel.rootAssembly
beamInstance = beamAssembly.Instance(name = 'Beam Instance',
               part = beamPart, dependent = ON)
```

En el código de Python primeramente se importa el módulo assembly, y posteriormente se crea la "Instance" dependiente con el nombre de `'Beam Instance'`.

4.5.2.4 Módulo STEP.

En este módulo indicamos a ABAQUS el tipo de análisis al que se va someter a nuestra estructura. Puesto que se trata de un análisis estático creamos un `StaticStep`.

```
#-------------------------------------------------------------------
#Create the Step

import step

#Create a static general step
beamModel.StaticStep(name = 'Loading Step', previous = 'Initial',
                     description = 'Cargas son aplicadas en la viga en este
                     Step')
```

En cuanto al código, el comando es muy sencillo e intuitivo. Se puede ver el nombre, el tipo de análisis, el step anterior que es el Inicial y una breve descripción del uso del Step.

4.5.2.5 Módulo INTERACTION

No hacemos referencia a este módulo en el código ya que se utiliza para establecer las relaciones entre varias partes creadas (en nuestro problema sólo existe una part).

4.5.2.6 Módulo LOAD

En este módulo se definen y aplican las cargas y las condiciones de contorno a nuestra estructura. A continuación se detalla cómo se establecen, las ya mencionadas varias veces en la memoria, cargas, apoyos y empotramientos.

```
#-------------------------------------------------------------------
#Apply loads

####Primero introduciremos una carga muy importante que es la carga de la
gravedad par que el calculo tenga en cuenta el peso de la viga.

beamModel.Gravity(comp2=-9.8, createStepName='Loading Step',
distributionType=UNIFORM, field='', name='Gravity')

import load

#Ponemos un punto en un lugar de la viga basandonos en nuestro conocimiento de
la geometria de la pieza
## Las cargas asi como los distintos apoyos no se puede poner en cualquier pun-
to que no sea un nodo si antes hacer una particion,
##primero diferenciaremos si la posicion en la que colocamos la carga es un
nodo u otro tramo de seccion para colocar directamente
##la carga o realizar una particion previa
##Hay que tener en cuenta que al realizar particiones tendremos mas elementos
en la malla.

LongitudSeccion=Longitud/float(NSecciones)
VectorPosicionesNodo=[]

while Contador0<=NSecciones:
      VectorPosicionesNodo.append(Vacio)
      VectorPosicionesNodo[Contador0]=Contador0*LongitudSeccion
      Contador0=Contador0+1

Contador0=0
Detecta=0
```

```
#Carga Puntual

for x in PosicionCargaPuntual:
      for j in VectorPosicionesNodo:
            if x==j:
                  Detecta=1

      if Detecta==0:
            beamPart.DatumPointByCoordinate(coords=(x,0,0))
            edges_for_CPuntual_Assignment=beamPart.edges.findAt((x,0,0))
            beamPart.PartitionEdgeByPoint(edge=edges_for_CPuntual_Assignment,
            point = (x,0,0))

      PuntoFuerzaPuntual=beamInstance.vertices.findAt(((x,0,0),),)
      beamModel.ConcentratedForce( name=NamesCargaPuntual[Contador0],
      createStepName='Loading Step',distributionType=UNIFORM,
      cf2=CargaPuntual[Contador0],
      region= (PuntoFuerzaPuntual,), field='', localCsys=None)
      Contador0=Contador0+1
      Detecta=0

Contador0=0
Detecta=0
```

En primer lugar le aplicamos la carga del peso propio de la viga con la carga denominada Gravity. En nuestro Script la complicidad de crear cargas en nuestro problema aumenta de forma significativa. En el cuadro superior se encuentra el código para crear cargas puntuales, es importante destacar que las cargas puntuales se aplican en nodos, y, los nodos que hay creados están situados en los extremos de la viga y un nodo cada `LongitudSeccion` por lo que si la variable de entrada que nos encontramos nos dice que la carga puntual hay que aplicarla en un lugar que no existe un nodo debemos crear ese nodo. El método para crear un nodo en el que se pueda aplicar una carga es mediante una partición que no afecte al comportamiento de la estructura, `PartitionEdgeByPoint`, y posteriormente buscar el punto en la "instance" y crear la carga. El código comienza importando el módulo LOAD, a continuación con un bucle "while" se crea un vector que guarda los valores de las posiciones de los nodos, `VectorPosicionesNodo`, utilizando la variable `LongitudSeccion`. Posteriormente, y teniendo en cuenta de que cabe la posibilidad de que exista más de una carga puntual, se inicia un bucle "for" que recorre el vector donde se encuentran los valores de las posiciones donde deben aplicarse las cargas puntuales, `PosicionCargaPuntual`, justo a continuación se inicia otro bucle "for" que recorrerá el vector que contiene las posiciones de los nodos, `VectorPosicionesNodo`. Si la posición en la que se encuentra una carga puntual coincide con algún nodo la variable `Detecta` asume el valor 1. De esta forma se sabe si es necesario crear un nodo nuevo o tenemos un nodo existente donde podemos aplicar la carga. El siguiente paso por tanto es comprobar el valor que tiene la variable `Detecta` iniciando una sentencia "if", que de no haber encontrado coincidencias tendrá el valor 0 y se procede a crear una partición:

1. Se crea un punto de información (Datum Point).
2. Se asigna el punto a una variable

3. Se crea la partición.

Una vez establecido el punto de aplicación de la carga utilizamos el comando `beamModel.ConcentratedForce` para aplicarla y detallamos la información antes descrita: Nombre, tipo, cantidad, dirección…

```
#Carga Lieal

for x in PosicionInicialCargaLineal:
      for j in VectorPosicionesNodo:
            if x==j:
                  Detecta=1

      for k in PosicionCargaPuntual:
            if x==k:
                  Detecta=1

      if Detecta==0:
            beamPart.DatumPointByCoordinate(coords=(x,0,0))
            edges_for_CLineal_Assignment=beamPart.edges.findAt((x,0,0))
            beamPart.PartitionEdgeByPoint(edge=edges_for_CLineal_Assignment,
            point = (x,0,0))

      Detecta=0

for x in PosicionFinalCargaLineal:
      for j in VectorPosicionesNodo:
            if x==j:
                  Detecta=1

      for k in PosicionCargaPuntual:
            if x==k:
                  Detecta=1

      if Detecta==0:
            beamPart.DatumPointByCoordinate(coords=(x,0,0))
            edges_for_CLineal_Assignment=beamPart.edges.findAt((x,0,0))
            beamPart.PartitionEdgeByPoint(edge=edges_for_CLineal_Assignment,
            point = (x,0,0))

      Detecta=0
```

Para la carga lineal se nos presentan dos nuevos problemas en el momento de programarla; por un lado tenemos dos variables de entrada para las posiciones, uno es la posición de inicio de la carga (`PosicionInicialCargaLineal)` y el otro es la posición final (`PosicionFinalCargaLineal`) y por otro lado ABAQUS en el momento de seleccionar donde aplicar la carga nos pide un punto de un tramo de sección, y no un nodo como ocurría con la carga puntual.

Primeramente se comprueba si las posiciones iniciales y finales de carga lineal coinciden con algún nodo anteriormente creado incluyendo los nodos que hemos creado para las cargas puntuales. Esta acción se realiza con el mismo mecanismo que para la carga lineal. Si no hay nodos que coincidan con las posiciones creamos

nuevos nodos pero esta vez con una finalidad distinta que en la carga lineal, ahora creando nuevos nodos conseguimos separar el tramo restante a la izquierda para la posición inicial y el tramo restante a la derecha para posición final donde no queremos que se aplique carga lineal.

```
##Uno todas las listas de puntos en una sola y las ordeno utilizando el comando
sort(), partiendo del primer punto hasta el ultimo
##voy cogiendo los puntos medios entre cada dos puntos y esa sera la region don-
de aplicaremos la carga distribuida.

ListadePun-
tos=VectorPosicionesNodo+PosicionCargaPuntual+PosicionInicialCargaLineal+Posicio
nFinalCargaLineal
ListadePuntos.sort()

##Como cabe la  posibilidad de que se repita algun punto en ListadePuntos debe-
mos eliminarlo, ya que de lo contrario tendriamos problemas en los
##siguientes pasos.

while len(ListadePuntos)>Contador1:
      if ListadePuntos[Contador0]==ListadePuntos[Contador1]:
            ListadePuntos.remove(ListadePuntos[Contador0])
      Contador0=Contador0+1
      Contador1=Contador1+1

Contador0=0
Contador1=1
Contador2=0
```

Una vez que se tienen todas las posiciones de los nodos se introducen en un único vector, `ListadePuntos`, y se colocan en orden de menor a mayor valor (izquierda a derecha) con el comando `ListadePuntos.sort()`.

Es posible que algunos de los puntos se repitan en el vector `ListadePuntos` por lo que es necesario eliminarlos para ello se ejecuta un bucle “while” que se repite tantas veces como el número de longitud del vector `ListadePuntos`. A continuación se ejecuta una sentencia “if” que elimina un valor si el siguiente es repetido.

```
for x in PosicionInicialCargaLineal:
      Region_Puntos=[]
      PosicionPrimerPunto=ListadePuntos.index(x)

      while (PosicionPrimerPunto+Contador0)<ListadePuntos.index
            (PosicionFinalCargaLineal[Contador2]):

            Region_Puntos.append(Vacio)

            Region_Puntos[Contador0]=ListadePuntos[PosicionPrimerPunto+
            Contador0]+ListadePuntos[PosicionPrimerPunto+Contador1]-
            ListadePuntos[PosicionPrimerPunto+Contador0])/float(2)

            Contador0=Contador0+1
            Contador1=Contador1+1

      Contador1=1

      for j in Region_Puntos:
            PuntoFuerzaLineal = beamInstance.edges.findAt(((j,0,0),),)
            Region_Carga = regionToolset.Region(edges = PuntoFuerzaLineal)
            beamModel.LineLoad(comp2=CargaLineal[Contador2], createStepName=
            'Loading Step', name=NamesCargaLineal[Contador2]+str(Contador1),
            region= Region_Carga)
            Contador1=Contador1+1

      Contador2=Contador2+1
      Contador0=0
      Contador1=1
```

Una vez tenemos las posiciones de los puntos ordenadas pasamos a hacer un vector nuevo donde guardaremos los puntos de los tramos donde vamos a aplicar la carga lineal, recordamos que la carga lineal se aplicaba en tramos y para ello se hacía referencia a un punto del tramo y no a nodos como ocurría con la carga puntual. Empezamos con un bucle “for” que recorre el vector `PosicionInicialCargaLineal` que contiene los puntos iniciales de las distintas cargas lineales y busca su posición en el vector `ListadePuntos` con el comando .index(). A continuación ejecutamos un bucle “while” que se ejecuta tantas veces como tramos haya desde el primer punto hasta el último dando al vector `Region_Puntos` los puntos medios de cada tramo.

Finalmente se ejecuta un nuevo bucle “for” que recorre el vector `Region_Puntos` y se asigna la misma a carga a cada tramo.

```
ListadePuntos=ListadePuntos+PosicionInicialCargaLineal+PosicionFinalCargaLineal

ListadePuntos.sort()

while len(ListadePuntos)>Contador1:
      if ListadePuntos[Contador0]==ListadePuntos[Contador1]:
            ListadePuntos.remove(ListadePuntos[Contador0])
      Contador0=Contador0+1
      Contador1=Contador1+1

Contador0=0
Contador1=1
```

Necesitamos incorporar a la lista de nodos los que corresponden a la posición inicial y final de la carga puntual para realizar el mismo proceso con el momento puntual.

```
#Momento Puntual

for x in PosicionMomentoPuntual:
      for j in ListadePuntos:
            if x==j:
                  Detecta=1

      if Detecta==0:
            beamPart.DatumPointByCoordinate(coords=(x,0,0))
            edges_for_MPuntual_Assignment=beamPart.edges.findAt((x,0,0))
            beamPart.PartitionEdgeByPoint(edge=edges_for_MPuntual_Assignment,
            point = (x,0,0))

      PuntoMomentoPuntual=beamInstance.vertices.findAt(((x,0,0),),)
      beamModel.Moment(cm3=MomentoPuntual[Contador0], createStepName='Loading
      Step', distributionType=UNIFORM, field='', localCsys=None,
      name=NamesMomentoPuntual[Contador0],
      region=(PuntoMomentoPuntual,))
      Contador0=Contador0+1
      Detecta=0

Contador0=0
Detecta=0
```

Para el momento puntual el procedimiento es idéntico que para carga puntual, ya que la carga se aplica en un nodo.

```
ListadePuntos=ListadePuntos+PosicionMomentoPuntual

ListadePuntos.sort()

while len(ListadePuntos)>Contador1:
      if ListadePuntos[Contador0]==ListadePuntos[Contador1]:
            ListadePuntos.remove(ListadePuntos[Contador0])
      Contador0=Contador0+1
      Contador1=Contador1+1

Contador0=0
Contador1=1
```

Nuevamente incorporamos al vector `ListadePuntos` las posiciones donde se aplica el momento puntual para hacer el mismo proceso con el apoyo simple.

```
#-----------------------------------------------------------------------
#Apply encastre

# Apoyo Simple
for x in PosicionApoyoSimple:
      for j in ListadePuntos:
            if x==j:
                  Detecta=1

      if Detecta==0:
            beamPart.DatumPointByCoordinate(coords=(x,0,0))
            edges_for_ASimple_Assignment=beamPart.edges.findAt((x,0,0))
            beamPart.PartitionEdgeByPoint(edge=edges_for_ASimple_Assignment,
            point = (x,0,0))

      PuntoApoyoSimple1 = beamInstance.vertices.findAt(((x,0,0),),)
      beamModel.PinnedBC( name = NamesApoyoSimple[Contador0], createStepName =
      'Initial', region = (PuntoApoyoSimple1,), )
      Contador0=Contador0+1
      Detecta=0

Contador0=0
Detecta=0
```

```
ListadePuntos=ListadePuntos+PosicionApoyoSimple

ListadePuntos.sort()

while len(ListadePuntos)>Contador1:
      if ListadePuntos[Contador0]==ListadePuntos[Contador1]:
            ListadePuntos.remove(ListadePuntos[Contador0])
      Contador0=Contador0+1
      Contador1=Contador1+1

Contador0=0
Contador1=1
```

```
#Encastre
for x in PosicionEncastre:
      for j in ListadePuntos:
            if x==j:
                  Detecta=1

      if Detecta==0:
            beamPart.DatumPointByCoordinate(coords=(x,0,0))
            edges_for_Encastre_Assignment=beamPart.edges.findAt((x,0,0))
            beamPart.PartitionEdgeByPoint(edge=edges_for_Encastre_Assignment,
            point = (x,0,0))

      PuntoEncastre = beamInstance.vertices.findAt(((x,0,0),),)
      beamModel.EncastreBC(name = NamesEncastre[Contador0],
      createStepName = 'Initial', region=(PuntoEncastre,),)
      Contador0=Contador0+1
      Detecta=0

Contador0=0
Detecta=0

ListadePuntos=ListadePuntos+PosicionEncastre

ListadePuntos.sort()

while len(ListadePuntos)>Contador1:
      if ListadePuntos[Contador0]==ListadePuntos[Contador1]:
            ListadePuntos.remove(ListadePuntos[Contador0])
      Contador0=Contador0+1
      Contador1=Contador1+1

Contador0=0
Contador1=1
```

Misma operación que para carga puntual y momento puntual son las que se realizan para Apoyo Simple y Empotramiento. Completamos el vector `ListadePuntos` con `PosicionEncastre` ya que la utilizaremos para el mallado en el siguiente módulo.

4.5.2.7 Módulo MESH.

Es en este módulo donde se definen la discretización mencionada en el apartado de elementos finitos, es decir, se determina la posición de los nodos. Decir que la herramienta se ha ejecutado utilizando una versión estudiante de ABAQUS la cual no permite discretizar la estructura superando un número máximo de nodos (1000), por lo que habría que tener cuidado al intentar ejecutar la herramienta introduciendo en la interfaz gráfica un alto grado de discretización. El valor introducido en el grado de discretización de la interfaz gráfica no coincide con el real que se aplica como veremos posteriormente en el código. Abaqus nos permite elegir que objeto discretizar, si la instance o la part.

La aplicación del mallado se puede ver en el último párrafo del cuadro de texto inferior, se selecciona el tipo de elemento que se quiere mallar, en este caso B21 que es la nomenclatura para referirse a elementos tipo Beam (barras reticuladas) y posteriormente se asigna la región que se quiere mallar. Finalmente se indica en cuántos elementos dividir la región seleccionada que es 4. Por lo tanto cuando indicamos en la interfaz gráfica un grado de discretización de 20 estamos diciendo que habrá 20 secciones distintas pero en realidad la estructura se dividirá en 4*20=80 nodos.

```
#Create the mesh
import mesh

ListaCarga =[]

while len(ListadePuntos)>Contador1:
      VectorNodo = ((ListadePuntos[Contador0],0,0),)
      ListaCarga.append(VectorNodo)
      VectorCarga =(((ListadePuntos[Contador1]-ListadePuntos[Contador2]/2)),0,0),
      ListaCarga.append(VectorCarga)
      Contador0=Contador0+1
      Contador1=Contador1+1


#Utilizamos ListaCarga para seleccionar toda la viga


beam_edges_for_meshing = beamInstance.edges.findAt(ListaCarga[Contador0])

VariableCarga=int(NSecciones)

for x in ListaCarga[1:(VariableCarga+1)]:
      beam_edges_for_meshing = beam_edges_for_meshing +
      beamInstance.edges.findAt(x)


beam_mesh_region = regionToolset.Region (edges = beam_edges_for_meshing)
beamElementType = mesh.ElemType(elemCode = B21, elemLibrary= STANDARD)
beamPart.setElementType(regions = beam_mesh_region, el-
emTypes=(beamElementType,))
beamPart.seedEdgeByNumber (edges = beam_edges_for_meshing, number = 4,)
beamPart.generateMesh()
```

Se genera un nuevo vector, `ListaCarga`, que hace referencia a todos los tramos y todos los "nodos" que tenemos a partir de `ListadePuntos` para poder asignar la región al mallado.

4.5.2.8 Módulo JOB

La función que se realiza en este módulo es crear un caso una vez definido todos los parámetros del problema y lanzarlo al procesador. A continuación se muestra la programación donde se mantienen los parámetros del Job que vienen predeterminados que son aconsejables para un análisis estático como es nuestro caso.

```
#-------------------------------------------------------------------
#Create and run the job
import job

mdb.Job(atTime=None, contactPrint=OFF, description='', echoPrint=OFF,
    explicitPrecision=SINGLE, getMemoryFromAnalysis=True, historyPrint=OFF,
    memory=50, memoryUnits=PERCENTAGE, model='Viga', modelPrint=OFF,
    name='BeamAnalysisJob', nodalOutputPrecision=SINGLE, queue=None, scratch=''
    , type=ANALYSIS, userSubroutine='', waitHours=0, waitMinutes=0)

mdb.jobs['BeamAnalysisJob'].submit(consistencyChecking=OFF)

#Do not reurn control till job is finished running

mdb.jobs['BeamAnalysisJob'].waitForCompletion()
```

4.5.2.9 Módulo VISUALIZATION

Visualización de los resultados. La programación que se muestra en el código muestra que se abre un archivo con extensión odb que es donde se encuentran guardados los resultados. Posteriormente le pedimos a ABAQUS que nos muestre la deformada de la viga que utilizaremos para hacernos una idea de los desplazamientos una vez hallamos obtenido una solución de la herramienta como se verá en el apartado de resolución de casos concretos.

En esta parte del script exportamos también los archivos con extensión rpt que contienen información sobre las tensiones y desplazamientos que se utilizaran para aplicar las restricciones al AG.

```
#-------------------------------------------------------------------------------
--------------------
#Visualization
import visualization
import odbAccess

beam_viewport = session.Viewport(name='Viewport: 1')
beam_Odb_Path = 'BeamAnalysisJob.odb'
an_odb_object = session.openOdb(name=beam_Odb_Path)
beam_viewport.setValues(displayedObject=an_odb_object)
beam_viewport.odbDisplay.display.setValues(plotState=(DEFORMED,))

odb = session.odbs['C:/Temp/BeamAnalysisJob.odb']
session.writeFieldReport(fileName='U2.rpt', append=ON,
        sortItem='Node Label', odb=odb, step=0, frame=1, outputPosition=NODAL,
        variable=(('U', NODAL, ((COMPONENT, 'U2'), )), ))

session.writeFieldReport(fileName='VonMises.rpt', append=ON,
        sortItem='Node Label', odb=odb, step=0, frame=1, outputPosition=NODAL,
              variable=(('S', INTEGRATION_POINT, ((INVARIANT, 'Mises'), )), ))
```

5 APLICACIÓN DE LA HERRAMIENTA A CASOS CONCRETOS Y RESULTADOS

En este apartado se han realizado distintos problemas utilizando la herramienta y se han comparado con cálculos analíticos. Puesto que para el cálculo de perfil nos encontramos con más de una variable (cada parámetro de la sección es una variable y no se utilizaran perfiles de vigas normalizados) en los cálculos analíticos se han realizado hipótesis que establecen relaciones entre los distintos parámetros de la sección para poder dar con una sección teórica aproximada que tendría la viga en caso de utilizar una única sección.

En conveniente indicar que las imágenes obtenidas de ABAQUS donde se muestra la deformada de las estructuras no son reales, pues los desplazamientos no superan los 3 milímetros. Éstas aparecen representadas a escala de forma que se incrementa el efecto de la deformación de forma que es mucho más intuitivo apreciar los desplazamientos.

5.1 CASO 1: VIGA EN MÉNSULA SOMETIDA A CARGA PUNTUAL EN SU EXTREMO DE SECCIÓN RECTANGULAR

5.1.1 RESULTADOS PROPORCIONADOS POR LA HERRAMIENTA

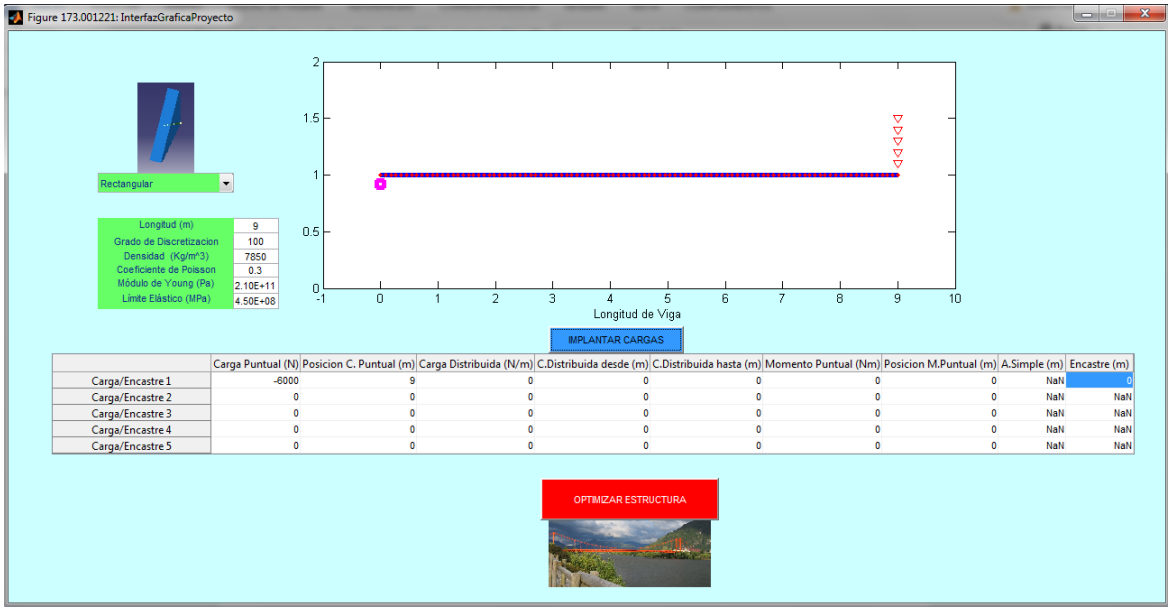


Figura 5.1.1.1: Diseño de la Viga en el Caso 1

Como se puede apreciar en la figura 5.1.1 se trata de optimizar la sección de una viga rectangular sometida a una carga de -6000N en un extremo con un empotramiento en el lado opuesto. Es muy importante destacar que este caso se ha realizado sin tener en cuenta el peso propio de la viga para el cálculo de tensión máxima y desplazamiento máximo, es decir, en ABAQUS no se estableció la gravedad. Los demás parámetros se muestran en la siguiente tabla:

PARÁMETROS GENERALES	
	Unidades en SI
Longitud	9
Nº Discretización	30
Densidad	7850
Módulo de Young	2,10E+11
Coeficiente de Poisson	0,3
Tipo de Perfil	Rectangular
Límite Elástico	4,50E+08

Tabla 5.1.1.1: Parámetros Generales de la viga

Los parámetros utilizados en los algoritmos genéticos son los establecidos en el script “AGsProyecto.m”. A continuación se muestra la solución:

Tiempo computacional = 7 horas

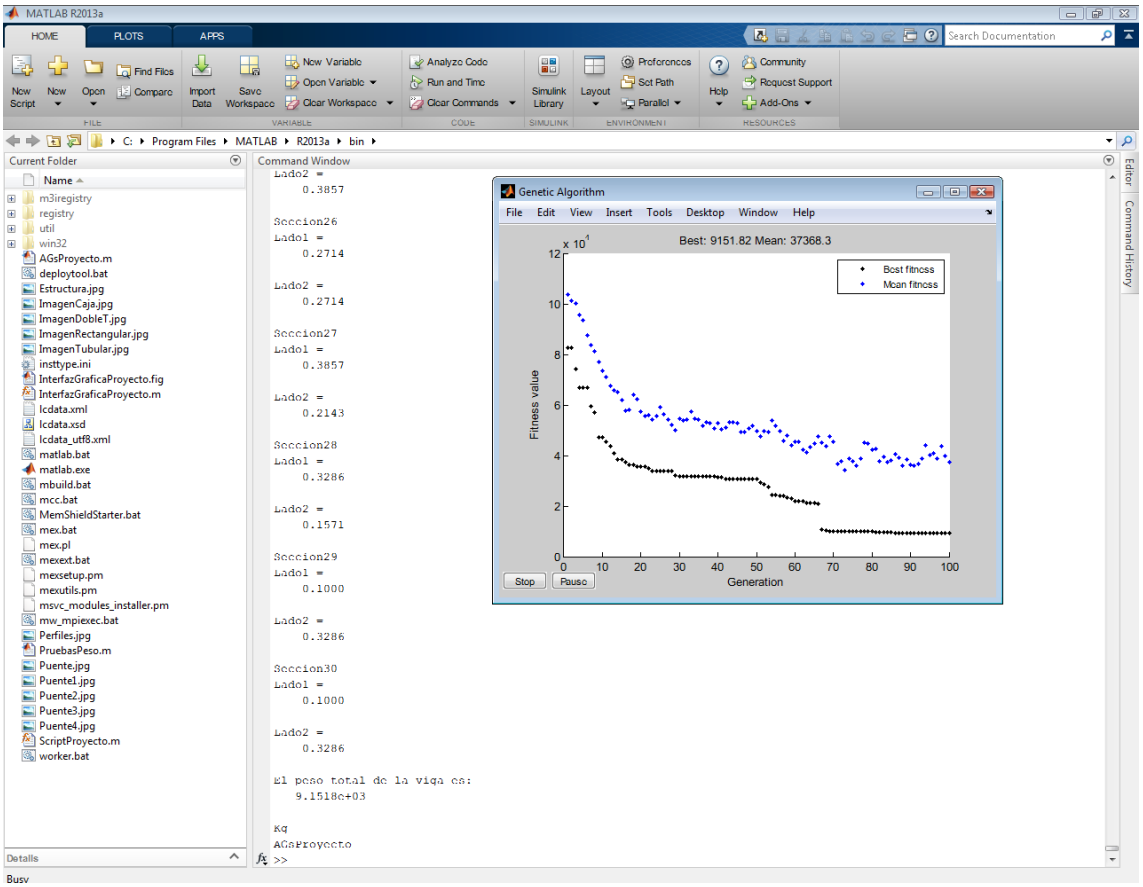


Figura 5.1.1.2: Gráfica Best Fitness y Masa Óptima de la Viga

Como se puede apreciar en la imagen la masa total de la viga es: 9151,82 Kilogramos.

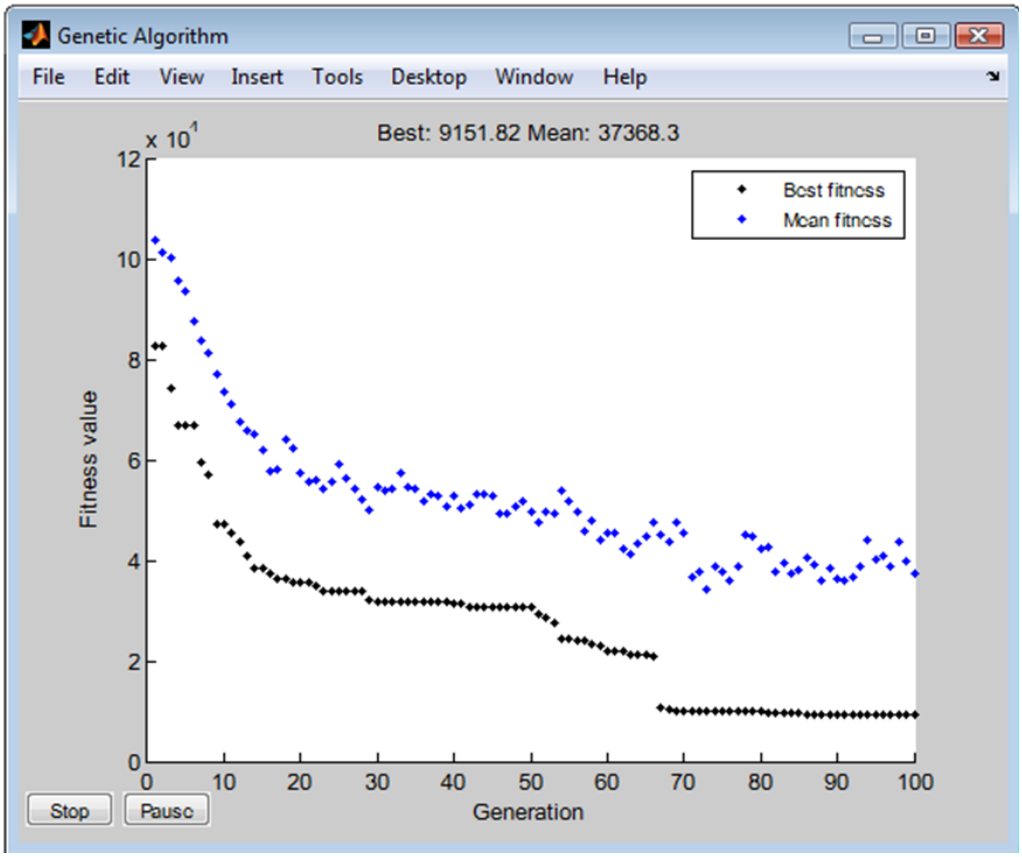


Figura 5.1.1.3: Gráfica Best Fitness Ampliada.

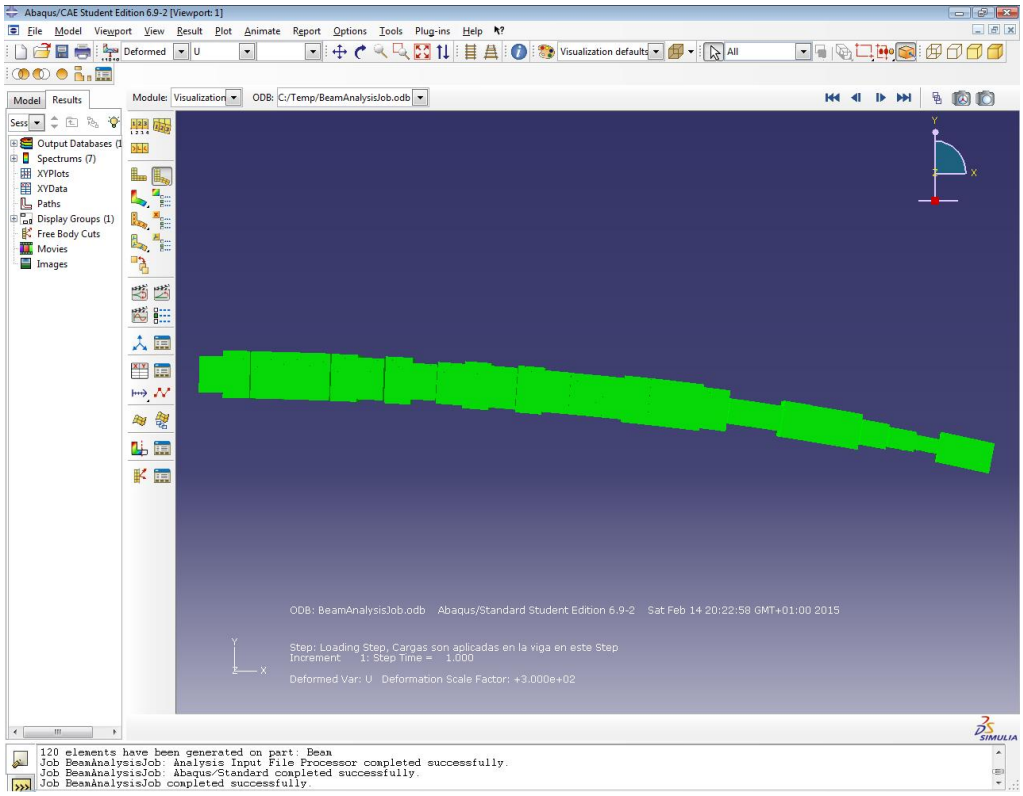


Figura 5.1.1.4: Deformada de la Viga en vista Frontal

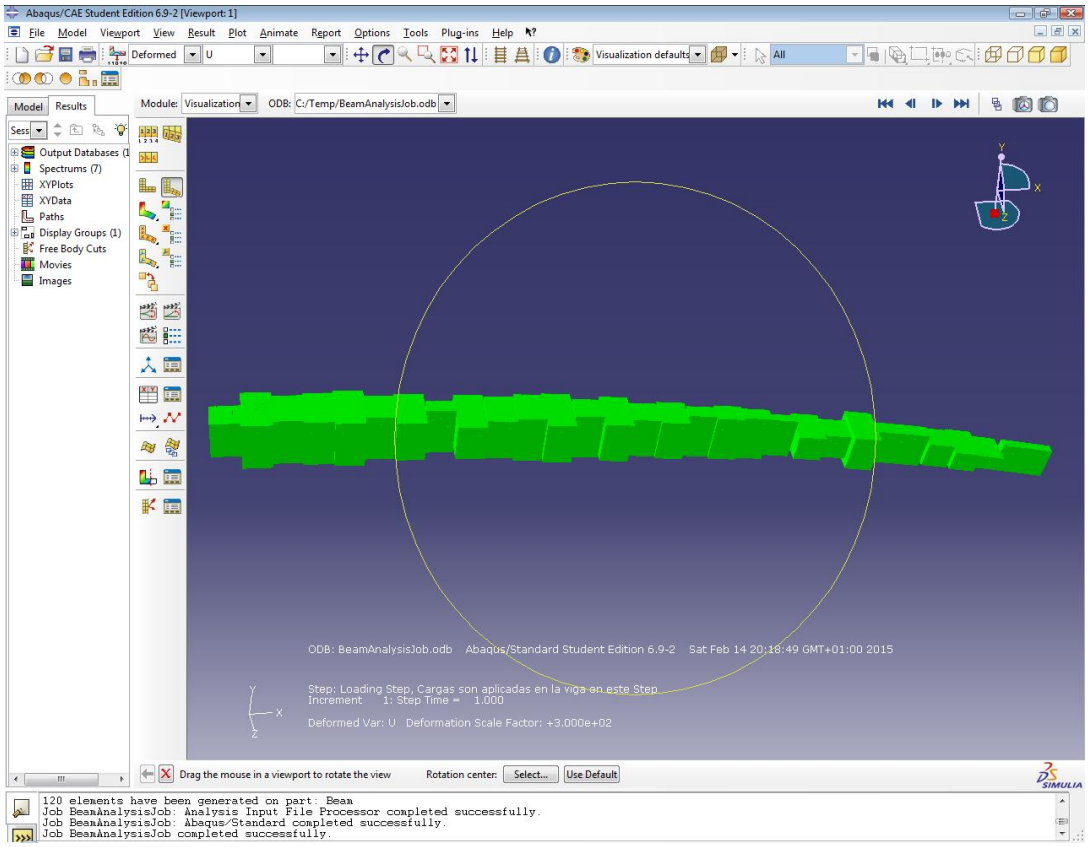


Figura 5.1.1.5: Deformada de la Viga en vista Oblicua.

Los datos correspondientes a la altura de cada sección son los siguientes obtenidos del archivo resultadosar.txt:

0.271400	0.500000	0.385700	0.442900	0.500000	0.328600	0.442900	0.157100
0.500000	0.385700	0.385700	0.500000	0.271429	0.442900	0.271429	0.328600
0.157100	0.214300	0.157100	0.100000	0.385714	0.271400	0.442857	0.157100
0.157100	0.271400	0.385700	0.328600	0.100000	0.100000		

Los datos correspondientes a la base de cada sección son los siguientes obtenidos del archivo resultadosar.txt:

0.385714	0.500000	0.500000	0.500000	0.500000	0.500000	0.442857	0.500000
0.385714	0.442857	0.500000	0.442857	0.500000	0.442857	0.442857	0.442857
0.500000	0.500000	0.500000	0.442857	0.271400	0.271400	0.385714	0.385714
0.385714	0.271400	0.214300	0.157100	0.328571	0.328571		

5.1.2 SOLUCIÓN TEÓRICA

Se realizará asignando un único tamaño de sección:

Al igual que en la herramienta tendremos en cuenta tanto la plasticidad del material como el desplazamiento máximo permitido que se ha establecido (que no supere los 0.003 metros). Calcularemos cuáles serían los parámetros de la sección óptimos en cuanto a masa que cumpla las condiciones:

5.1.2.1 Plastificación del Material

La sección mínima que tendría el material sin que este llegase a plastificar la calcularemos teniendo en cuenta la Ley de Navier: "El momento flector produce tensión normal en la sección transversal ya que tracciona unas fibras y comprime las del lado contrario. Por ese motivo la tensión normal variará con la posición en la que nos encontremos en la pieza prismática". Al ser una sección rectangular la tensión máxima se producirá en los puntos que estén situados más alejados del eje neutro, (el eje neutro se encuentra en y=0). Por lo que la tensión normal será tensión principal ya que en los extremos del rectángulo (se supondrá cuadrado para el cálculo de la sección) la tensión tangencial será nula. La tensión de Von Mises será por tanto la tensión normal en dichos puntos.

Sabemos que en una viga con todas las cargas en el plano la tensión normal a la que está sometido el material en un punto será:

$$\sigma_z = \frac{M_x}{Ix} y$$

M_x antihorario en eje x positivo.

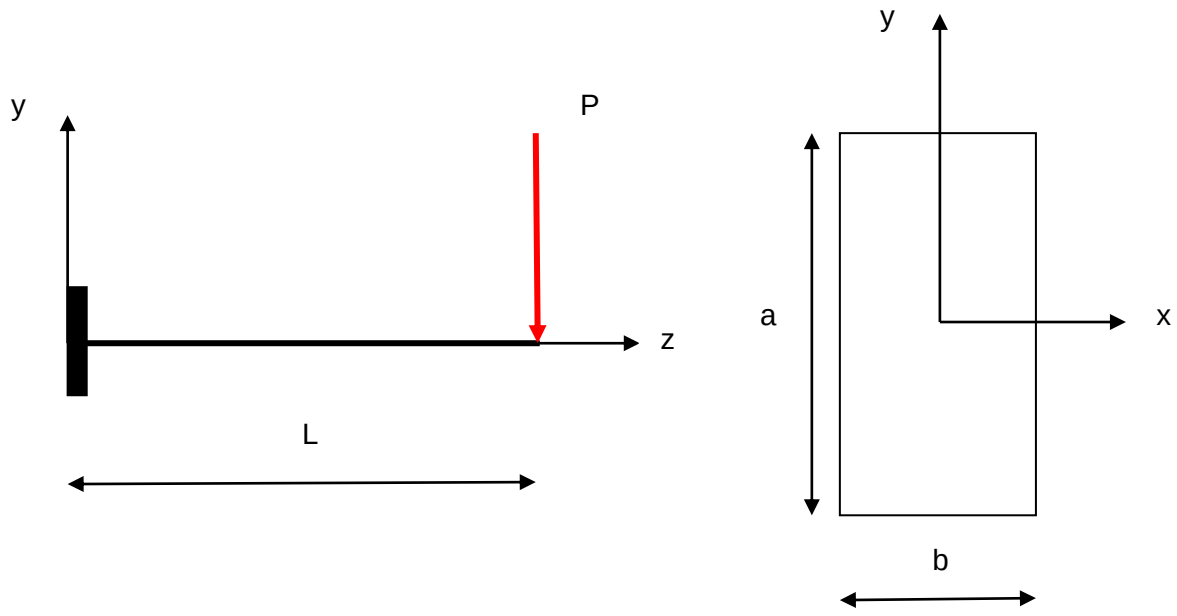


Figura 5.1.2.1.1: Representación Gráfica del Problema

La tensión máxima a la que puede estar sometido el material es el límite elásticao, σ_y. Los puntos con mayor tensión serán los que estén más alejados del eje neutro (eje neutro tensión normal es igual a 0) por lo tanto teniendo en cuenta que el valor máximo que puede adoptar los lados es de a metros, la y_{max} será a/2 metros. El tramo de viga que estará sometido a mayores esfuerzos flectores será el empotramiento donde el momento será P·L. El momento de Inercia de un rectángulo respecto del eje x es $I_x = (b \cdot a^3)/12$.

$M_{empotramiento}$ = P·L = 6000N·9m=54000 N·m

σ_y = 4,50E+08 Pa

y = a/2 m

A continuación realizaremos dos hipótesis distintas para el cálculo de la masa:

- Si consideramos una sección cuadrada tendremos:

$$\sigma_y = \frac{M_x}{Ix} y \rightarrow a = b = \sqrt[4]{\frac{12 \cdot M_x \cdot y}{\sigma_y}} = \sqrt[3]{\frac{12 \cdot 54000}{4.5E+08 \cdot 2}} = 0{,}09m$$

Por lo tanto la masa de la viga sería:

M= A*L*ρ = $0{,}16^2$·9·7850 = 763,02 Kg

- Si consideramos que a adoptará el valor máximo (a=0.5), favoreciendo la resistencia de la estructura:

$$b = \frac{12 \cdot 54000 \cdot 0{,}25}{(4.5E+08) \cdot 0{,}5^3} = 0{,}00288\, m$$

Por lo tanto la masa de la viga sería:

M= A*L* ρ = 0,5·0,00288·9·7850 = 115 Kg

Habrá que calcular también la masa de la viga teniendo en cuenta el desplazaminto máximo y ver cuál es más restrictiva.

5.1.2.2 Desplazamiento Máximo Permitido

Como tenemos una viga empotrada en un extremo y carga en el opuesto deducimos que el desplazamiento máximo se producirá en el extremo donde se encuentra aplicada la carga. Calcularemos el desplazamiento máximo utilizando el Teorema de Mohr. Podría emplearse el método de elementos finitos, en cambio como se ha comentado anteriormente es un cálculo que proporciona una aproximación de la solución. Esto se debe a que se discretiza los elementos. El Teorema de Morh es una simplificación de las leyes de Navier-Bresse para el cálculo de movimientos en piezas prismáticas donde se realiza la integral de la ley de momentos flectores y en el teorema de Mohr la sintetiza en sumatorio de áreas del diagrama de momentos flectores como puede verse a continuación. De esta forma obtenemos un resultado

más exacto que por MEF haciendo referencia a los infinitos puntos que componen la viga.

$$\downarrow V_B = V_{A+}\theta_A \cdot d_{A\rightarrow ejeVB} + \sum_{A}^{B}\left[\frac{(\acute{A}rea\ Mf)_{A\rightarrow Bi}}{EI} d_{CDG\rightarrow ejeVB}\right]$$

En el empotramiento (punto A) los giros y desplazamientos son nulos.

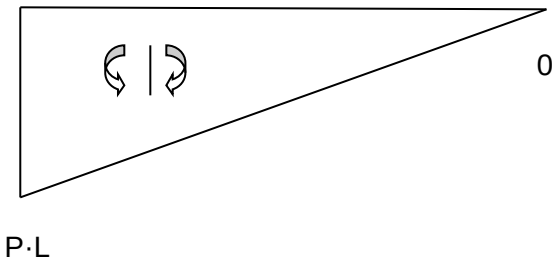


Figura 5.1.2.2.1: Diagrama de Momentos Flectores

Simplificando nos queda que la flecha máxima en el punto donde se encuentra aplicada la carga es:

$$\text{Flecha máxima} = 0{,}003\ = \frac{\mathrm{PL}^3}{3\mathrm{EI}}$$

Si consideramos una sección cuadrada tendremos:

$$a = b = \sqrt[4]{\frac{6000 \cdot 12 \cdot 9^3}{3 \cdot 2{,}1e11 \cdot 0{,}003}} = 0{,}41$$

Por lo tanto la masa de la viga sería:

M= A*L* ρ = $0{,}41^2 \cdot 9 \cdot 7850$ = 11876.3 Kg

Si consideramos que a adoptará el valor máximo (a=0.5), favoreciendo la resistencia de la estructura:

$$b = \frac{6000 \cdot 12 \cdot 9^3}{3 \cdot 2{,}1e11 \cdot 0{,}003 \cdot 0{,}5^3} = 0{,}2222m$$

Por lo tanto la masa de la viga sería:

$$M = A^{*}L^{*}\rho = 0{,}5 \cdot 0{,}2222 \cdot 9 \cdot 7850 = 7844.2 \text{ Kg}$$

Como dato de referencia hemos de quedarnos con el que obtendríamos si la viga fuese de sección cuadrada, ya que al algoritmo le damos libertad para que varíe ambos lados del rectángulo, si quisiéramos compararle con el valor que obtendríamos de dejar constante la altura deberíamos hacer los mismo para el algoritmo. Entre los obtenidos por sección cuadrada nos quedamos con el más restrictivo, es decir, con el calculado teniendo en cuenta el desplazamiento máximo.

El resultado final que nos da la herramienta es una masa de 9151,82 kilogramos, que comparada con la que obtendríamos con sección constante (11876.3 Kg) es mucho menor. El resultado se mejoraría si se realizasen más iteraciones y aumentáramos el grado de discretización pero se aumentaría de forma considerable el tiempo computacional.

5.2 CASO 2: VIGA BIAPOYADA DE SECCIÓN DOBLE T SOMETIDA A CARGA DISTRIBUIDA

5.2.1 RESULTADOS PROPORCIONADOS POR LA HERRAMIENTA

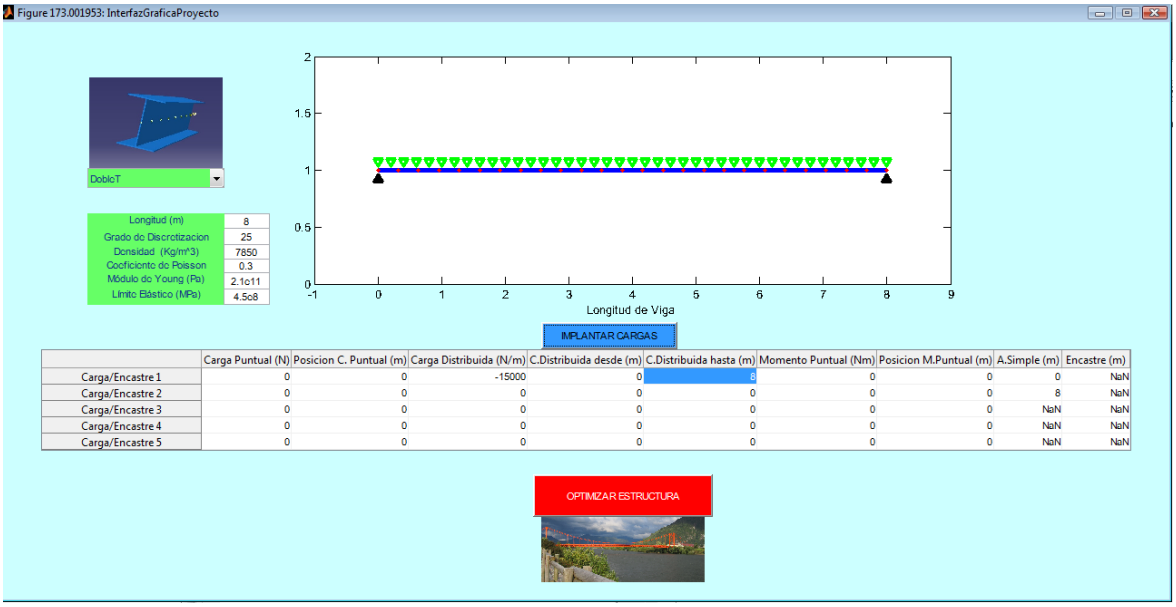


Figura 5.2.1.1: Diseño de la Viga en el Caso 2

Como se puede apreciar en la figura 5.2.1 se trata de optimizar la sección de una viga DobleT sometida a una carga distribuida de -15000N/m a lo largo de toda su longitud. Los demás parámetros se muestran en la siguiente tabla:

PARÁMETROS GENERALES	
	Unidades en SI
Longitud	8
Nº Discretización	25
Densidad	7850
Módulo de Young	2,10E+11
Coeficiente de Poisson	0,3
Tipo de Perfil	DobleT
Límite Elástico	4,50E+08

Tabla 5.2.1.1: Parámetros Generales de la viga

Los parámetros utilizados en los algoritmos genéticos son los establecidos en el script “AGsProyecto.m”. A continuación se muestra la solución:

Tiempo computacional = 4 horas

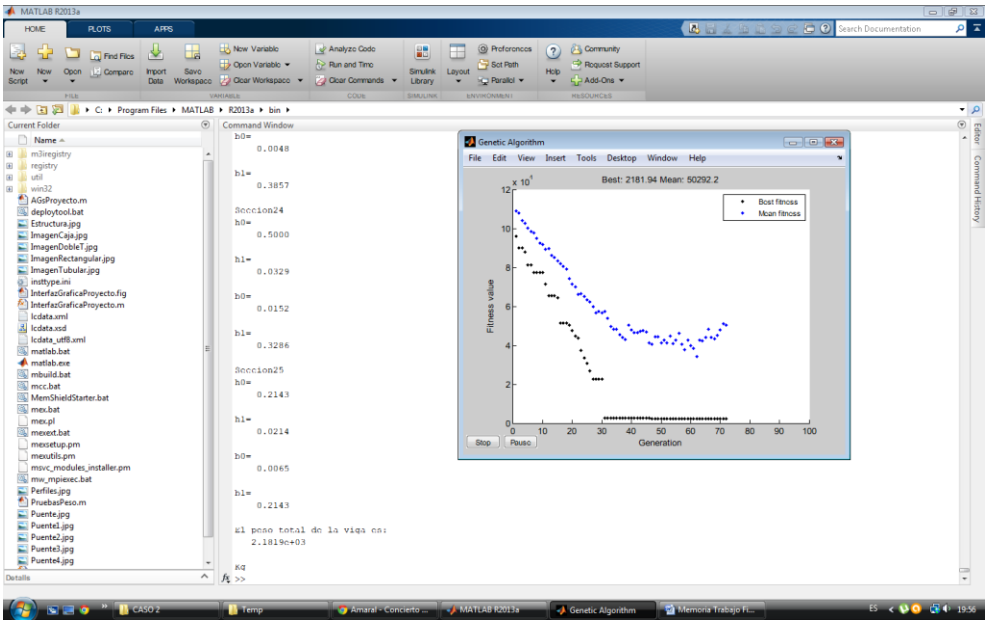


Figura 5.2.1.2: Gráfica Best Fitness y Peso Óptimo de la Viga

Como se puede apreciar en la imagen la masa total de la viga es 2181,9 Kilogramos.

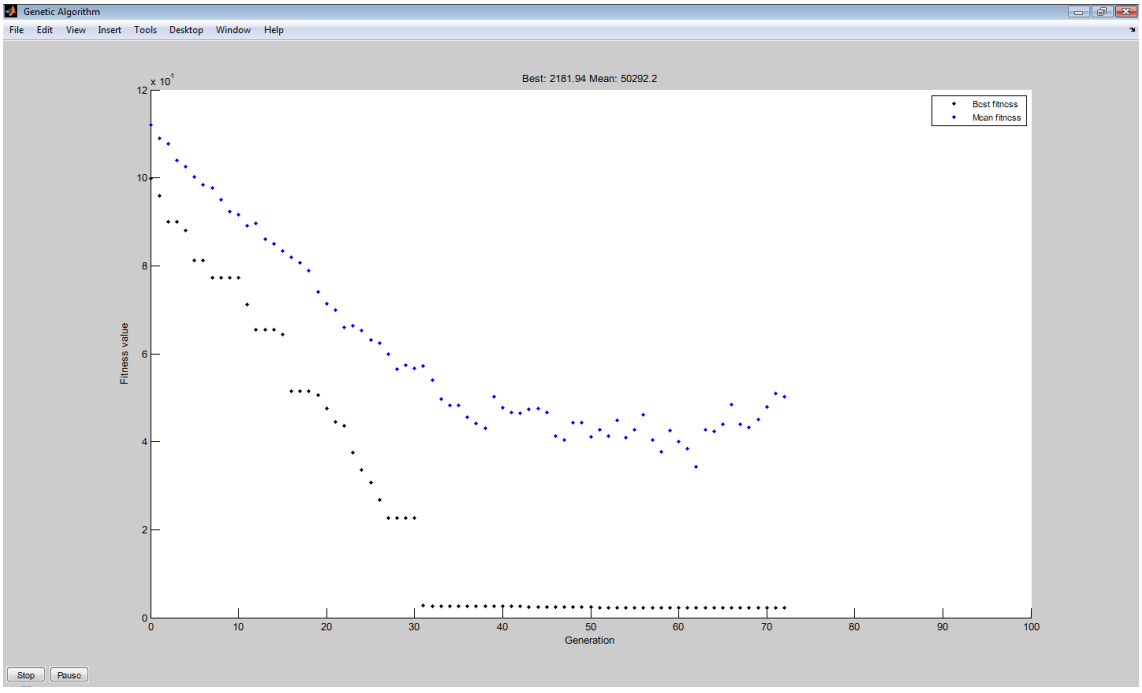


Figura 5.2.1.3: Gráfica Best Fitness Ampliada.

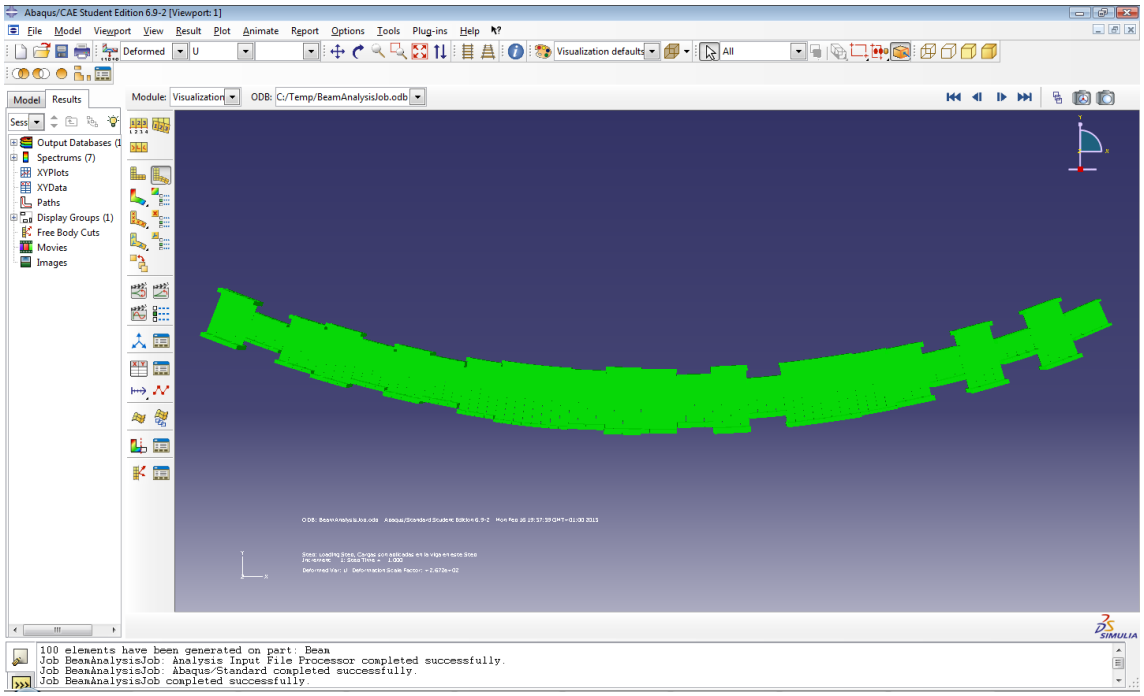


Figura 5.2.1.4: Deformada de la Viga en vista Frontal

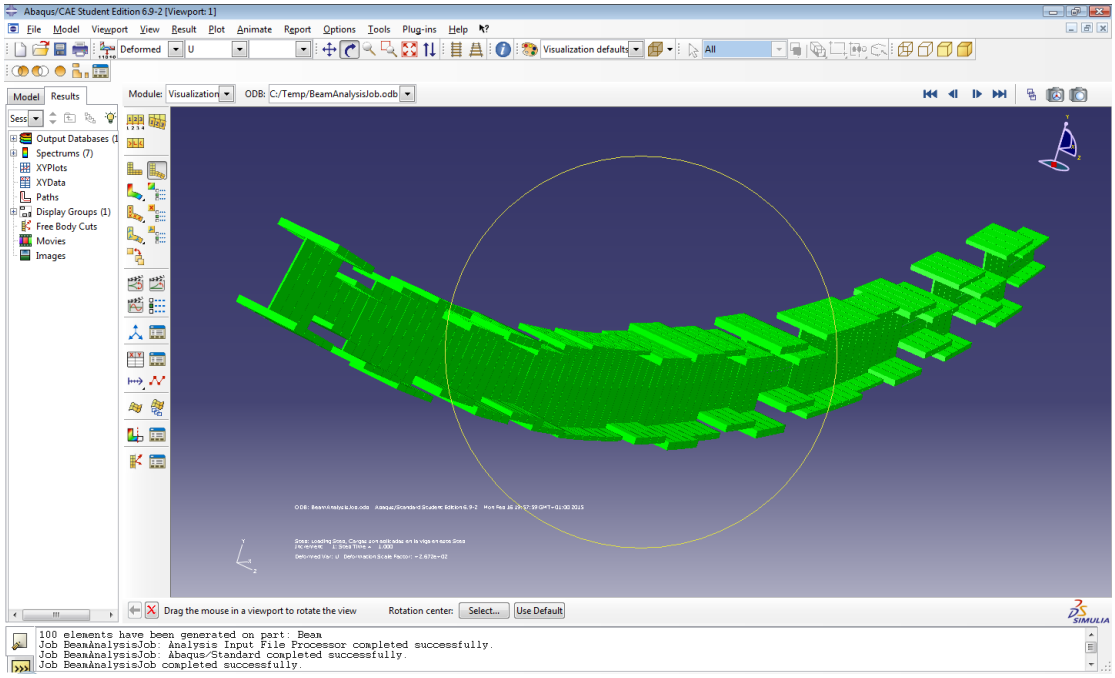


Figura 5.2.1.5: Deformada de la Viga en Vista Oblicua.

Se puede observar como los mayores tamaño de sección se establecen alrededor de la zona central donde la resistencia es mayor.

Los datos correspondientes a la altura de las almas de cada sección son los siguientes obtenidos del archivo resultadosh0.txt:

0.385714	0.214286	0.385714	0.442857	0.328571	0.385714	0.328571	0.442857
0.500000	0.500000	0.442857	0.500000	0.500000	0.385714	0.500000	0.271429
0.500000	0.500000	0.500000	0.442857	0.214286	0.500000	0.157143	0.500000
0.214286							

Los datos correspondientes al ancho de las almas de cada sección son los siguientes obtenidos del archivo resultadosb0.txt:

0.011688	0.006494	0.011688	0.013420	0.009957	0.011688	0.009957	0.013420
0.015152	0.015152	0.013420	0.015152	0.015152	0.011688	0.015152	0.008225
0.015152	0.015152	0.015152	0.013420	0.006494	0.015152	0.004762	0.015152
0.006494							

Los datos correspondientes a la altura de las alas de cada sección son los siguientes obtenidos del archivo resultadosh0.txt:

0.038571	0.027143	0.032857	0.032857	0.038571	0.044286	0.044286	0.032857
0.027143	0.027143	0.044286	0.050000	0.038571	0.044286	0.050000	0.044286
0.038571	0.038571	0.027143	0.044286	0.044286	0.038571	0.038571	0.032857
0.021429							

Los datos correspondientes al ancho de las alas de cada sección son los siguientes obtenidos del archivo resultadosb1.txt:

0.385714	0.271429	0.328571	0.328571	0.385714	0.442857	0.442857	0.328571
0.271429	0.271429	0.442857	0.500000	0.385714	0.442857	0.500000	0.442857
0.385714	0.385714	0.271429	0.442857	0.442857	0.385714	0.385714	0.328571
0.214286							

5.2.1 SOLUCIÓN TEÓRICA

Se realizará asignando un único tamaño de sección:

Al igual que en la herramienta tendremos en cuenta tanto la plasticidad del material como el desplazamiento máximo permitido que se ha establecido (que no supere los 0.003 metros). Calcularemos cuáles serían los parámetros de la sección óptimos en cuanto a masa que cumpla las condiciones:

5.2.1.1 Plastificación del Material

La sección mínima que tendría el material sin que este llegase a plastificar la calcularemos teniendo en cuenta la Ley de Navier. Puesto que la tensión máxima a la que está sometida la viga será en el punto medio solamente tendremos tensiones normales ya que el cortante en el punto medio de la viga es nulo.

Sabemos que en una viga con todas las cargas en el plano la tensión normal a la que está sometido el material en un punto será:

$$\sigma_z = \frac{M_x}{Ix} y$$

M_x antihorario en eje x positivo.

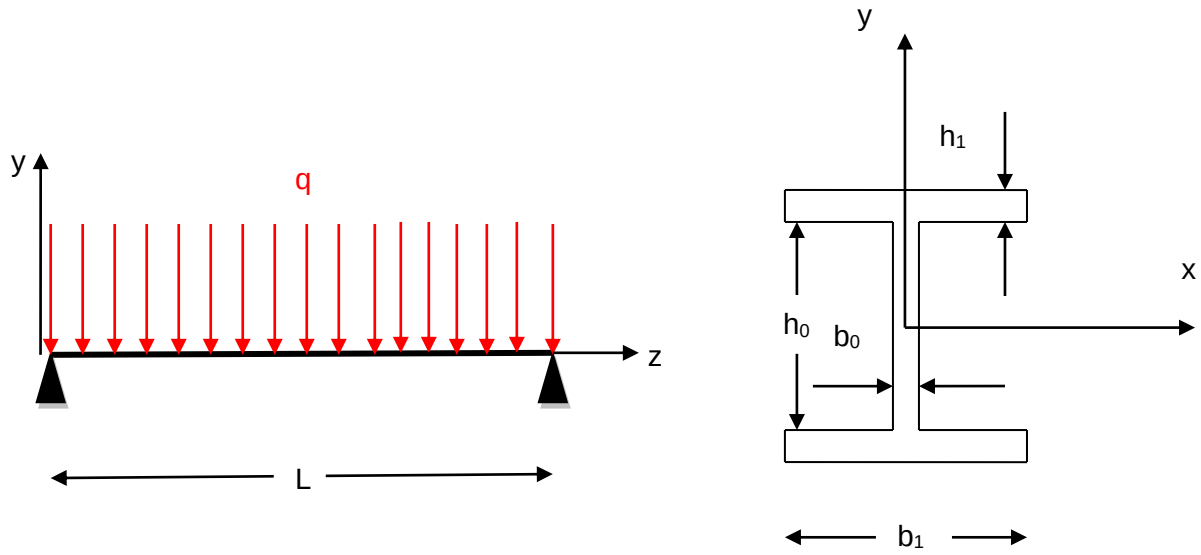


Figura 5.2.1.1.1: Representación Gráfica del Problema

La relación existente entre los parámetros es:

$$b_0 = \frac{h_0}{33} \qquad b1 = 10 \cdot h_1$$

La tensión máxima a la que puede estar sometido el material es el límite elásticao, σ_y. Los puntos de la sección que estarán sometidos a mayores tensiones estarán situados a una distancia de $h_0/2+ h_1$. En el punto más alejado del eje neutro. Por lo que la y_{max} será $h_0/2+ h_1$.El tramo de viga que estará sometido a mayores esfuerzos flectores será el centro de la viga donde el momento será (q·L^2)/8. El momento de Inercia de la sección respecto del eje x es $I_x = I_{x1} + I_{x2} + I_{x3}$. Para el cálculo de momento de Inercia de I_{x1} y I_{x3} aplicaremos el teorema de ejes paralelos.

$$I_{x1} = I_{x3} = \frac{h_1^3 \cdot b_1}{12} + h_1 \cdot b_1 \left(\frac{h_1}{2} + \frac{h_0}{2}\right)^2$$

$$I_{x3} = \frac{h_0^3 \cdot b_0}{12}$$

Simplificando para tener sólo dos parámetros:

$$I_{x1} = I_{x3} = \frac{5 \cdot h_1^4}{6} + 10 \cdot h_1^2 \cdot \left(\frac{h_1}{2} + \frac{h_0}{2}\right)^2$$

$$I_{x3} = \frac{h_0^4}{366}$$

El momento de inercia total será la suma de los momentos de inercia:

$$I_x = \frac{h_0^4}{366} + 2 \cdot \left(\frac{5 \cdot h_1^4}{6} + 10 \cdot h_1^2 \cdot \left(\frac{h_1}{2} + \frac{h_0}{2}\right)^2\right)$$

El momento en el punto medio de la sección añadiendo a la carga una aproximación del peso propio de la viga de 500N/m tenemos:

$M_{puntomedio}$ = (q·L^2)/8= (15500·8^2)/8=124000 N·m

σ_y = 4,50E+08 Pa

y = h_0/2+ h_1= h_0

Si establecemos una relación de $h_0 = 10 \cdot h_1$ como hipótesis para el cálculo de la sección tendremos que:

$$I_x = \frac{h_0^4 \cdot 5801}{91500}$$

y = h_0/2+ h_1= 3h_0/5

$$\sigma_y = \frac{M_x}{Ix} y \rightarrow h_0 = \sqrt[4]{\frac{12 \cdot M_x \cdot y}{\sigma_y}} = \sqrt[3]{\frac{91500 \cdot 124000 \cdot 3}{4.5E+08 \cdot 5801 \cdot 5}} = 0{,}13m$$

Por lo tanto la masa de la viga sería:

M= A*L* ρ = 237,96Kg

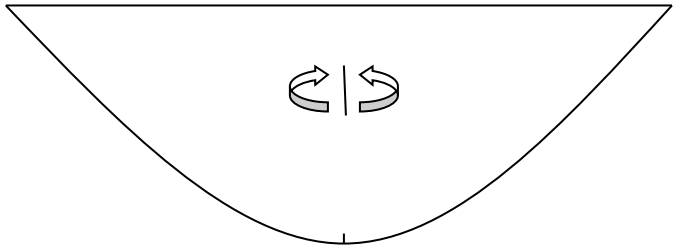
5.2.1.2 Desplazamiento Máximo Permitido

Como tenemos una viga biapoyada con carga distribuida a lo largo de toda la longitud deducimos que el desplazamiento máximo se producirá en el centro de la viga

(L/2) ya que es donde el momento flector es máximo. Calcularemos el desplazamiento máximo utilizando el Teorema de Mohr:

$$\downarrow V_B = \downarrow V_{A+}\theta_A \cdot d_{A\rightarrow ejeVB} + \sum_{A}^{B}\left[\frac{(Área\ Mf)_{A\rightarrow Bi}}{EI} d_{CDG\rightarrow ejeVB}\right]$$

Aplicando la fórmula entre los dos apoyos obtenemos que el giro en el punto A es (q·L^3)/(24·E·I).



$$\frac{q \cdot L^2}{8}$$

Figura 5.2.1.2.1: Diagrama de Momentos Flectores.

$$\downarrow V_A = 0 + \frac{qL^2}{8} \cdot \frac{L}{2} - \frac{2 \cdot BaseParábola \cdot AlturaParábola/3}{E \cdot I} \cdot \frac{3 \cdot BaseParábola}{8}$$

Simplificando nos queda que la flecha máxima en el punto donde se encuentra aplicada la carga es:

$$\text{Flecha máxima} = 0{,}003\ = \frac{5\text{qL}^4}{384\text{EI}}$$

Si realizamos la misma hipótesis que para calcular la sección última de plastificación tenemos, añadiendo a los 15000N/m una aproximación del peso final que tendrá la viga de 2500N/m ya que a la hora de realizar el cálculo se tiene en cuenta el peso propio tenemos:

$$h_0 = \sqrt[4]{\frac{5 \cdot 17500 \cdot 91500 \cdot 8^4}{384 \cdot 2{,}1e11 \cdot 0{,}003 \cdot 5801}} = 0{,}39$$

Por lo tanto la masa de la viga sería:

$M = A*L*\rho = 2199{,}83$ Kg

Como podemos ver el resultado obtenido con la herramieta es tan sólo ligeramente menor que el obtenido de forma analítica para una sóla sección. Esto es debido a que nos hemos visto obligados a reducir las tolerancias de los parámetros para reducir el tiempo computacional.

5.3 CASO 3: VIGA EN MÉNSULA DE SECCIÓN CAJÓN SOMETIDA A CARGA DISTRIBUIDA

5.3.1 RESULTADOS PROPORCIONADOS POR LA HERRAMIENTA

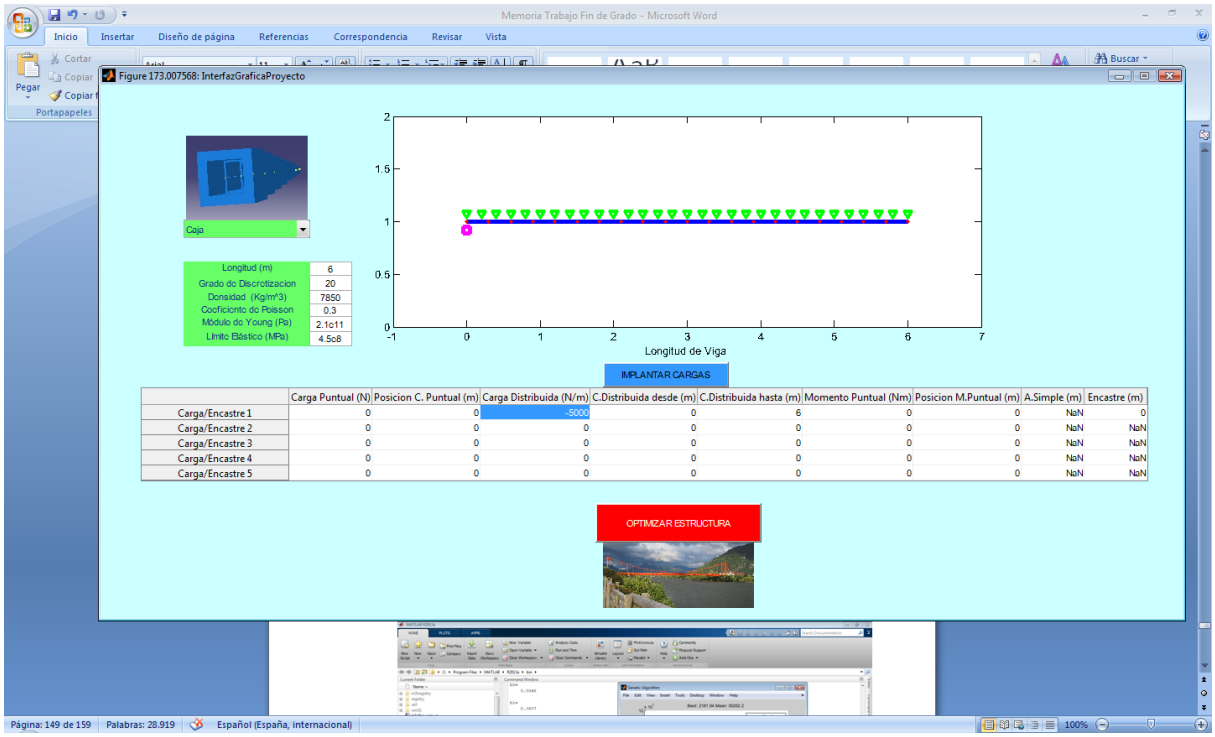


Figura 5.3.1.1: Diseño de la Viga en el Caso 3

Como se puede apreciar en la figura 5.3.1 se trata de optimizar la sección de una viga con sección de caja sometida a una carga distribuida de -5000N/m a lo largo de toda su longitud. Los demás parámetros se muestran en la siguiente tabla:

PARÁMETROS GENERALES	
	Unidades en SI
Longitud	6
Nº Discretización	20
Densidad	7850
Módulo de Young	2,10E+11
Coeficiente de Poisson	0,3
Tipo de Perfil	Caja
Límite Elástico	4,50E+08

Tabla 5.3.1.1: Parámetros Generales de la viga

Los parámetros utilizados en los algoritmos genéticos son los establecidos en el script “AGsProyecto.m”. A continuación se muestra la solución:

Tiempo computacional = 7 horas

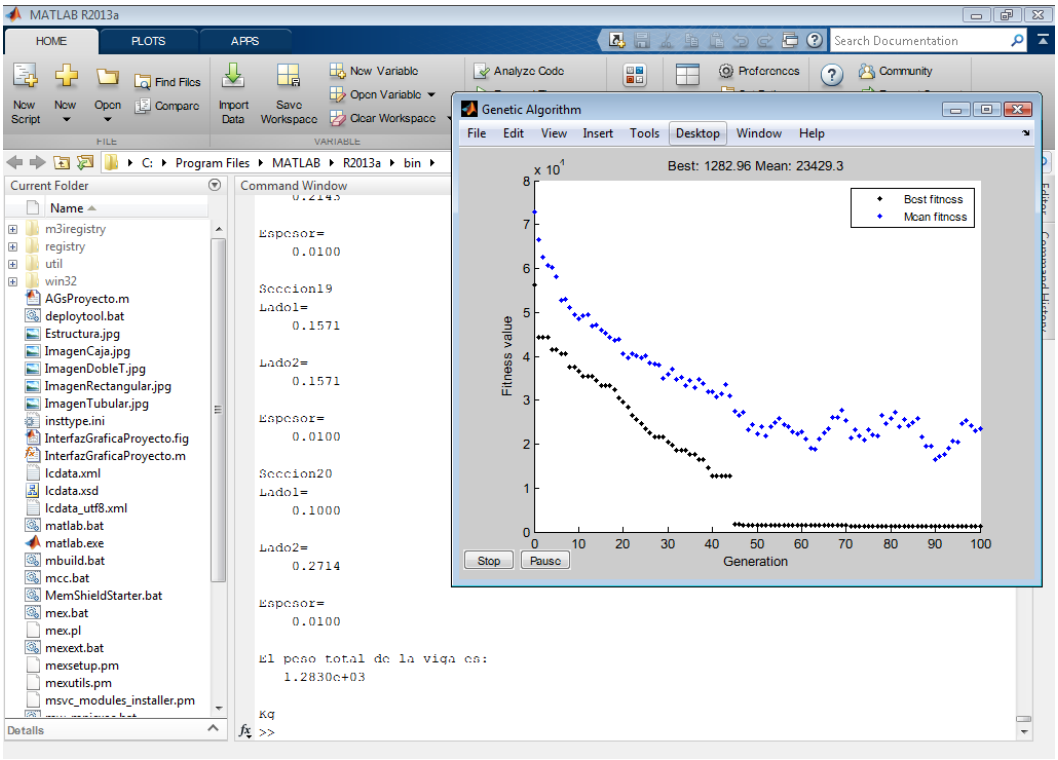


Figura 5.3.1.2: Gráfica Best Fitness y Peso Óptimo de la Viga

Como se puede apreciar en la imagen el peso total de la viga es 1283 Kg

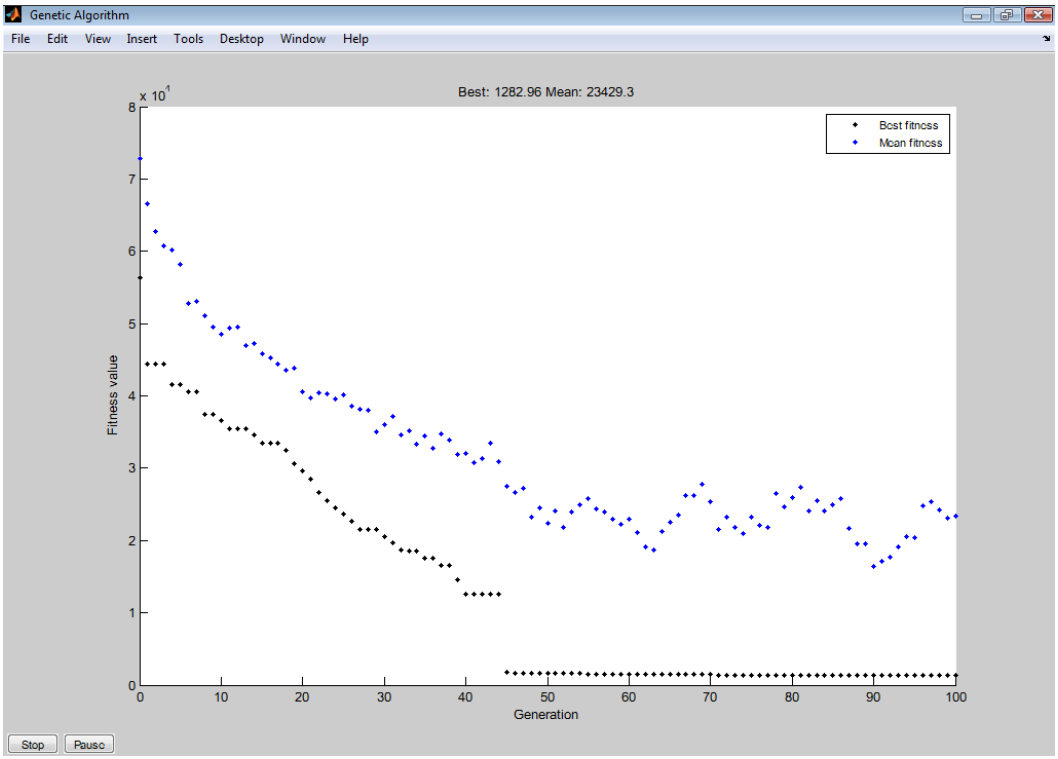


Figura 5.3.1.3: Gráfica Best Fitness y Peso Óptimo de la Viga Ampliada

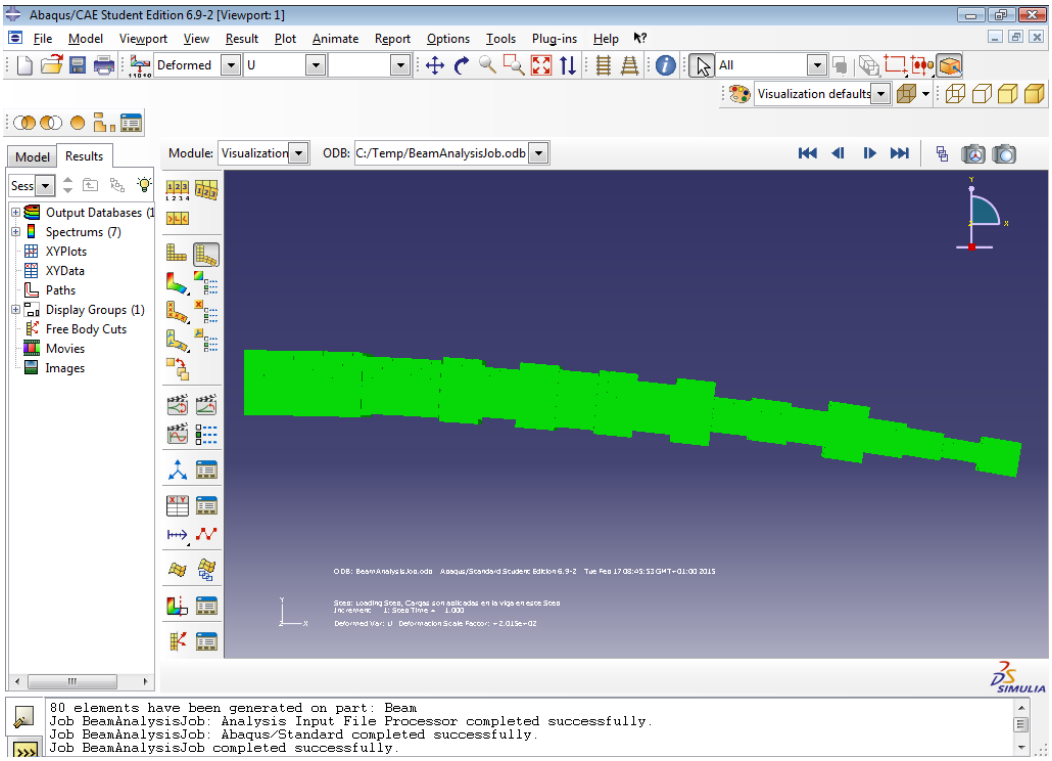


Figura 5.3.1.4: Deformada de la Viga en vista Frontal

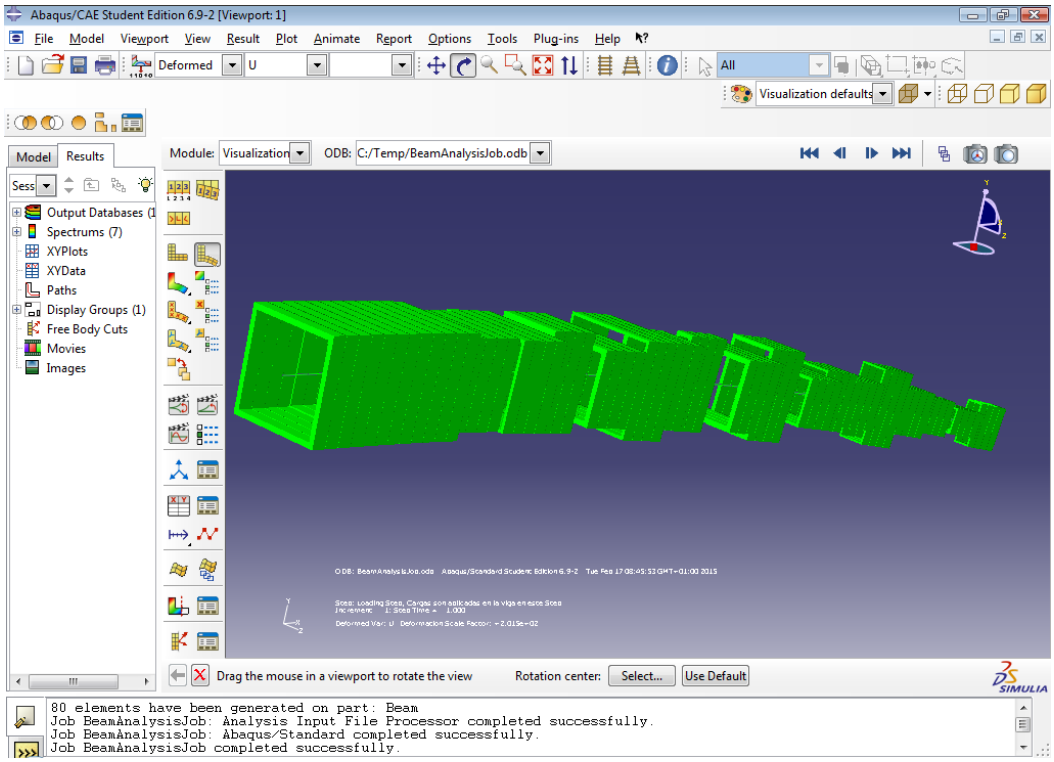


Figura 5.3.1.5: Deformada de la Viga en Vista Oblicua

Los datos correspondientes a la altura de cada sección son los siguientes obtenidos del archivo resultadosac.txt:

```
 0.500000  0.500000  0.500000  0.442857  0.500000  0.328571  0.500000  0.500000
0.271429  0.100000  0.442857  0.271429  0.442857  0.214286  0.214286  0.157143
0.157143  0.100000  0.157143  0.100000
```

Los datos correspondientes a la base de cada sección son los siguientes obtenidos del archivo resultadosbc.txt:

```
 0.500000  0.500000  0.500000  0.442857  0.442857  0.500000  0.385714  0.500000
0.442857  0.500000  0.385714  0.500000  0.271429  0.328571  0.271429  0.442857
0.271429  0.214286  0.157143  0.271429
```

Los datos correspondientes al espesor de cada sección son los siguientes obtenidos del archivo resultadostc.txt:

0.026714 0.032286 0.015571 0.021143 0.015571 0.032286 0.026714 0.015571 0.026714 0.021143 0.026714 0.021143 0.015571 0.010000 0.015571 0.010000 0.010000 0.010000 0.010000 0.010000

5.3.1 SOLUCIÓN TEÓRICA

Se realizará asignando un único tamaño de sección:

Al igual que en la herramienta tendremos en cuenta tanto la plasticidad del material como el desplazamiento máximo permitido que se ha establecido (que no supere los 0.003 metros). Calcularemos cuáles serían los parámetros de la sección óptimos en cuanto a masa que cumpla las condiciones:

5.3.1.1 Plastificación Del Material

La sección mínima que tendría el material sin que este llegase a plastificar la calcularemos teniendo en cuenta la Ley de Navier. Sabemos que en una viga con todas las cargas en el plano la tensión normal a la que está sometido el material en un punto será:

$$\sigma_z = \frac{M_x}{Ix} y$$

M_x antihorario en eje x positivo.

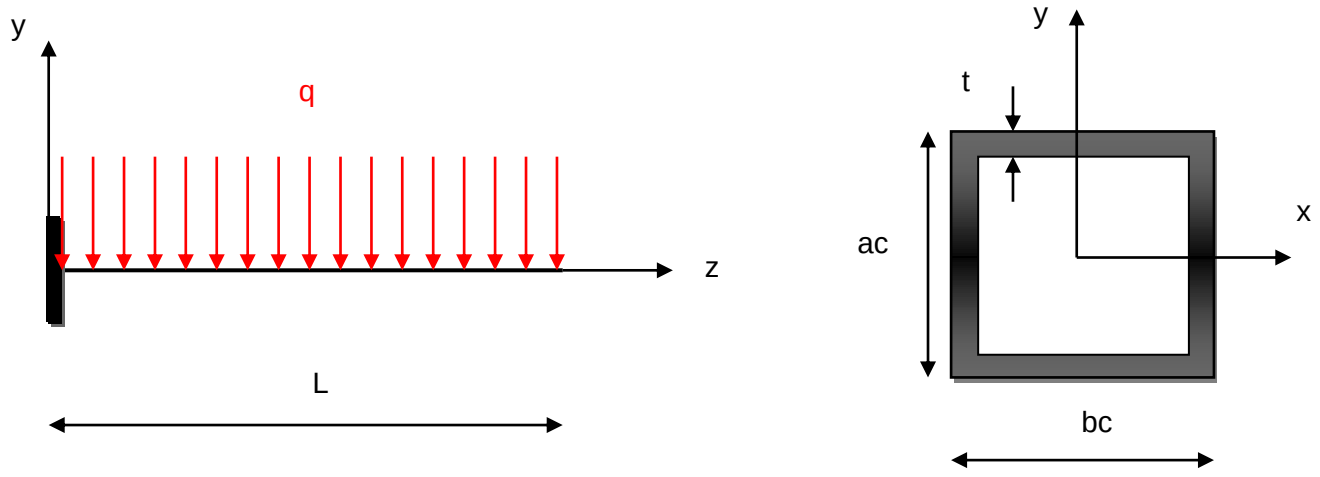


Figura 5.3.1.1.1: Representación Gráfica del Problema

La tensión máxima a la que puede estar sometido el material es el límite elásticao, σ_y. Los puntos con mayor tensión serán los que estén más alejados del eje neutro (eje neutro tensión normal es igual a 0) por lo tanto teniendo en cuenta que el valor máximo que puede adoptar los lados es de ac metros, la y_{max} será ac/2 metros. El tramo de viga que estará sometido a mayores esfuerzos flectores será el

empotramiento donde el momento será $q \cdot L^2/2$. El momento de Inercia de un rectángulo hueco respecto del eje x es $I_x = I_{x1} - I_{x2} = (bc \cdot ac^3)/12 - ((bc-tc) \cdot (ac-tc)^3)/12$

$M_{empotramiento} = q \cdot L^2/2 = 5000 \cdot 6^2/2 = 90000$ N·m

σ_y = 4,50E+08 Pa

y = ac/2 m

Si consideramos una sección cuadrada (ac=bc), con un espesor tc = ac/10 y despreciando el peso propio de la viga tendremos:

$$\sigma_y = \frac{M_x}{Ix} y \rightarrow ac = bc = \sqrt[4]{\frac{12 \cdot M_x \cdot y}{\sigma_y}} = \sqrt[3]{\frac{12 \cdot 90000}{4.5E+08 \cdot 0{,}0286 \cdot 2}} = 0{,}34m$$

Por lo tanto la masa de la viga sería:

M= A*L* ρ =1034,5 Kg

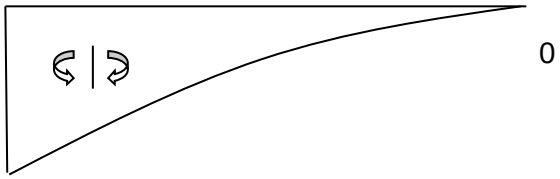
Habrá que calcular también la masa de la viga teniendo en cuenta el desplazaminto máximo y ver cuál es más restrictiva.

5.3.1.2 Desplazamiento Máximo Permitido

Como tenemos una viga empotrada en un extremo con carga distribuida a lo largo de toda la longitud, deducimos que el desplazamiento máximo se producirá en el extremo donde se encuentra aplicada la carga. Calcularemos el desplazamiento máximo utilizando el Teorema de Mohr:

$$\downarrow V_B = V_{A+}\theta_A \cdot d_{A \rightarrow ejeVB} + \sum_{A}^{B} \left[\frac{(\text{Área } Mf)_{A \rightarrow Bi}}{EI} d_{CDG \rightarrow ejeVB} \right]$$

En el empotramiento (punto A) los giros y desplazamientos son nulos.



$q \cdot L^2/2$

Figura 5.3.1.2.1: Diagrama de Momentos Flectores

Simplificando la ecuación del Teorema de Mohr nos queda que la flecha máxima de la viga en el extremo libre es:

$$\text{Flecha máxima} = 0{,}003 \ = \frac{\mathrm{qL}^4}{8\mathrm{EI}}$$

Si consideramos una sección cuadrada (ac=bc), con un espesor tc = ac/5 y despreciando el peso propio de la viga tendremos:

$$ac = bc = \sqrt[4]{\frac{5000 \cdot 6^4}{8 \cdot 2{,}1e11 \cdot 0{,}003 \cdot 0{,}0286}} = 0{,}46m$$

Por lo tanto la masa de la viga sería:

M= A*L* ρ = 1893,6 Kg

El resultado final que nos da la herramienta es un peso de 1283 kilogramos que comparado con el que obtendríamos con sección constante (1893,6 Kg) es mucho menor. El resultado se mejoraría si se realizasen más iteraciones y aumentáramos el grado de discretización pero se aumentaría de forma considerable el tiempo computacional.

5.4 CASO 4: VIGA CONTINUA TUBULAR DE DOS VANOS CON APOYOS SIMPLES SOMETIDA A CARGA PUNTUAL EN EL SEGUNDO VANO

5.4.1 RESULTADO PROPORCIONADO POR LA HERRAMIENTA

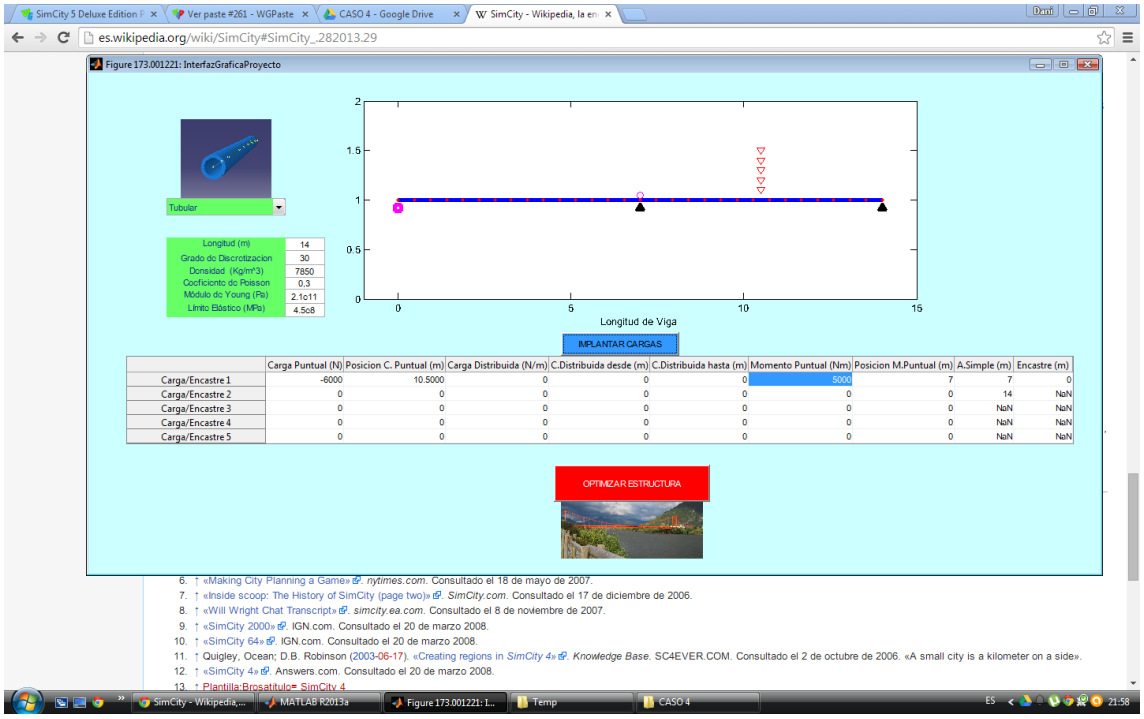


Figura 5.4.1.1: Diseño de la Viga en el Caso 3

Como se puede apreciar en la figura 5.4.1 se trata de optimizar la sección de una viga con sección tubular sometida a una carga puntual de valor 6000N aplicada en el punto medio del segundo tramo y a un momento puntual de valor 5000N/m aplicado en el punto medio de la viga coincidiendo con un apoyo simple. Los demás parámetros se muestran en la siguiente tabla:

PARÁMETROS GENERALES	
	Unidades en SI
Longitud	14
Nº Discretización	30
Densidad	7850
Módulo de Young	2,10E+11
Coeficiente de Poisson	0,3
Tipo de Perfil	Tubular
Límite Elástico	4,50E+08

Tabla 5.4.1.2: Parámetros Generales de la viga

Los parámetros utilizados en los algoritmos genéticos son los establecidos en el script “AGsProyecto.m”. A continuación se muestra la solución:

Tiempo computacional = 9 horas

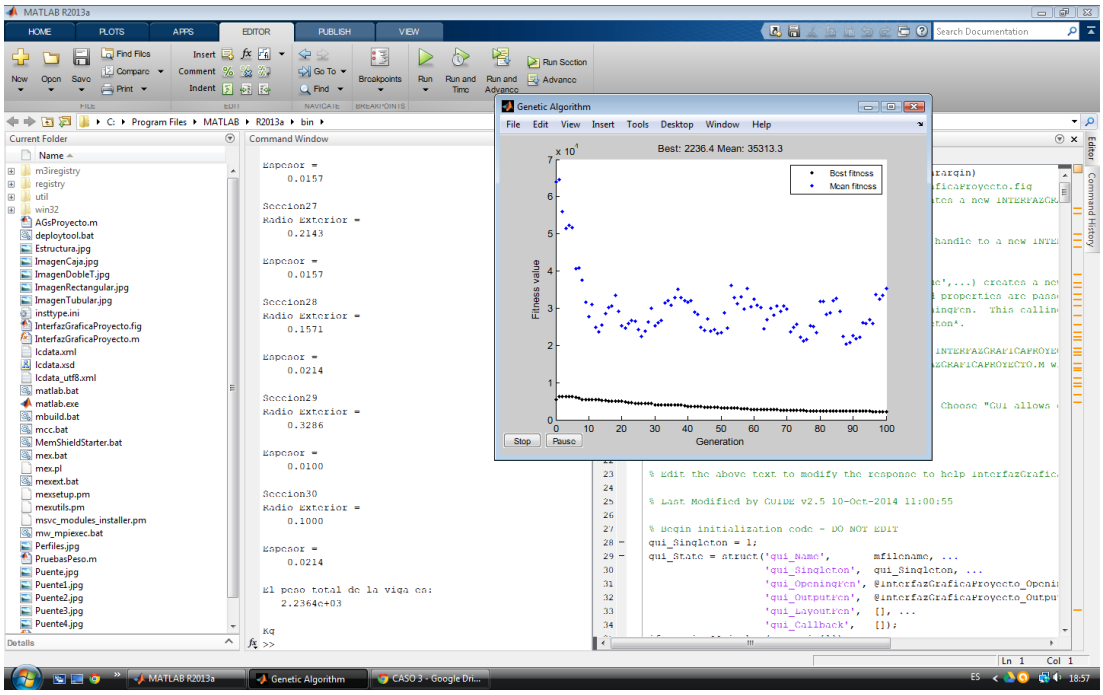


Figura 5.4.1.2: Gráfica Best Fitness y Peso Óptimo de la Viga

Como se puede apreciar en la imagen el peso total de la viga es 1283 Kg

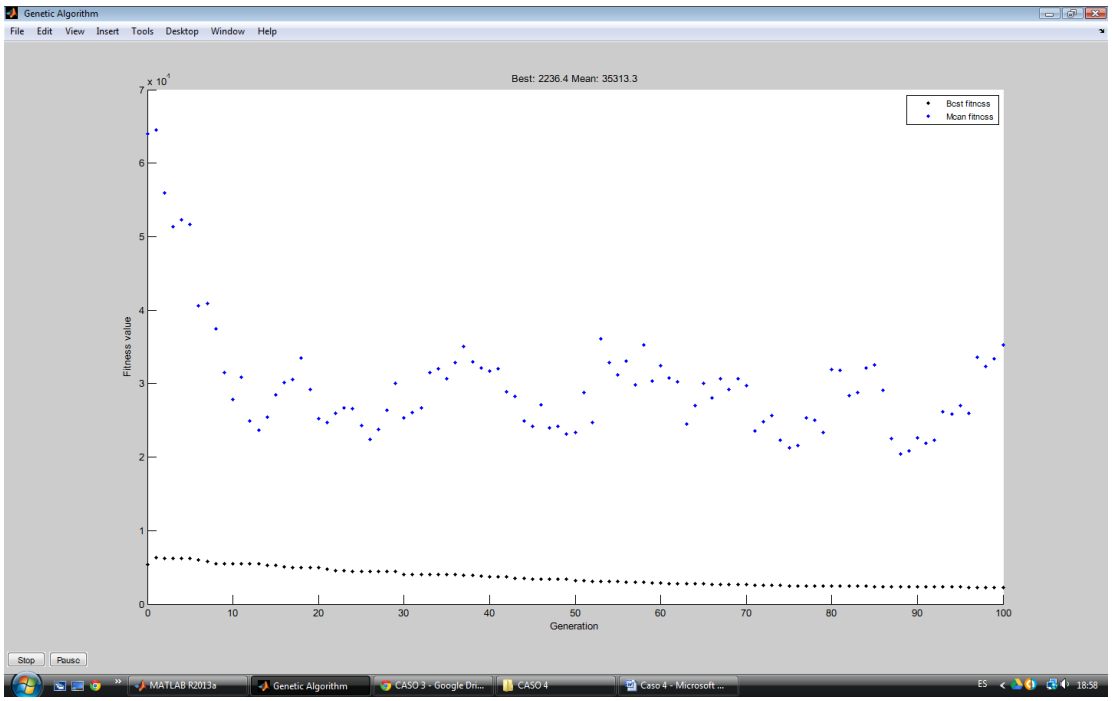


Figura 5.4.1.3: Gráfica Best Fitness y Peso Óptimo de la Viga Ampliada

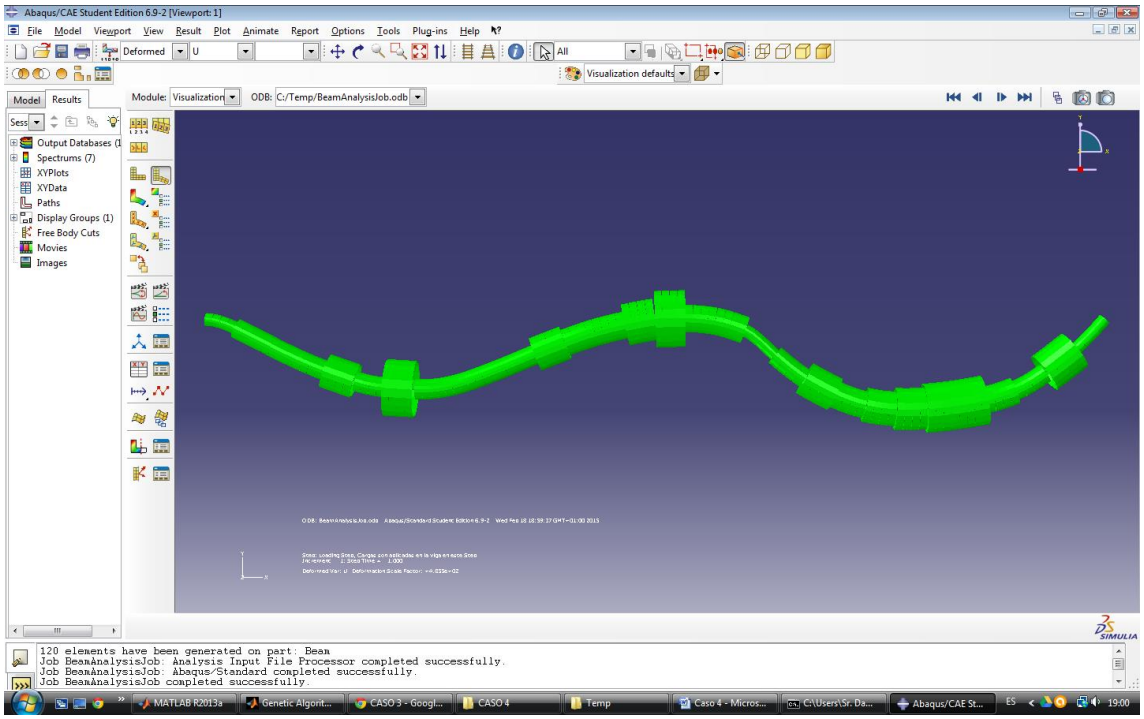


Figura 5.4.1.4: Deformada de la Viga en vista Frontal

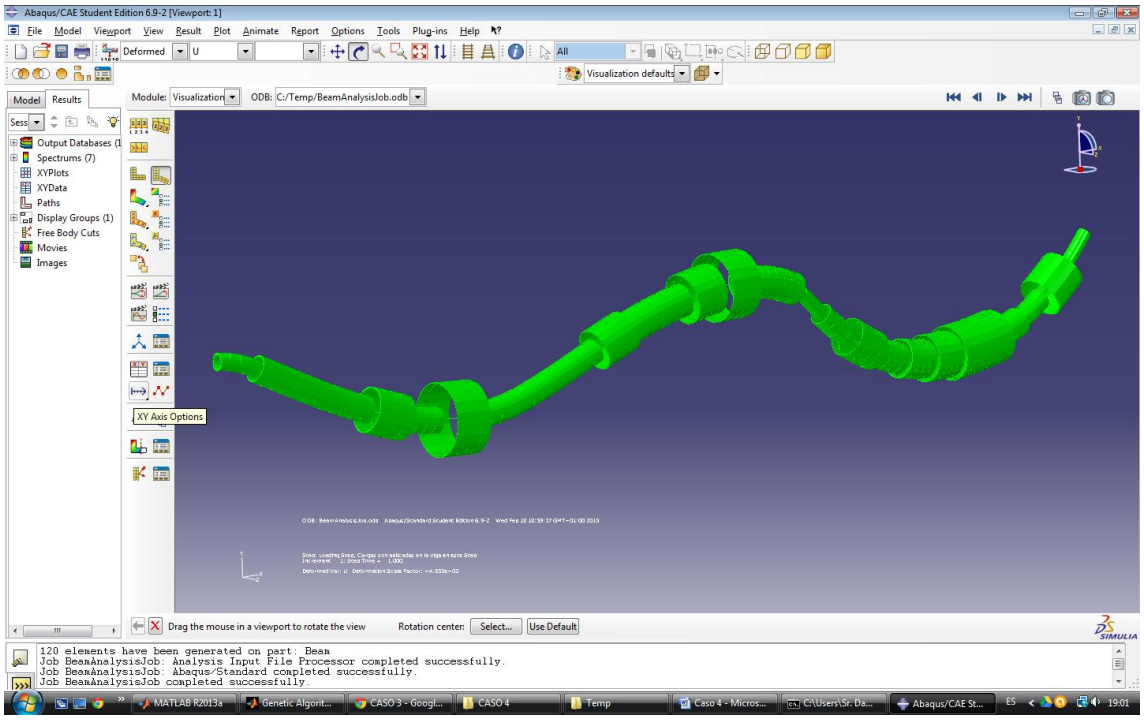


Figura 5.4.1.5: Deformada de la Viga en Vista Oblicua

Los datos correspondientes al radio exterior de cada sección son los siguientes obtenidos del archivo resultadosrs.txt:

```
 0.100000  0.157143  0.157143  0.157143  0.271429  0.157143  0.442857  0.157143
0.157143  0.157143  0.157143  0.271429  0.214286  0.214286  0.328571  0.442857
0.214286  0.214286  0.100000  0.157143  0.271429  0.214286  0.271429  0.328571
0.385714  0.385714  0.214286  0.157143  0.328571  0.100000
```

Los datos correspondientes al espesor de cada sección son los siguientes obtenidos del archivo resultadosts.txt:

```
 0.021429  0.015714  0.015714  0.021429  0.015714  0.010000  0.010000  0.010000
0.010000  0.044286  0.015714  0.015714  0.015714  0.038571  0.010000  0.010000
0.021429  0.010000  0.010000  0.010000  0.010000  0.015714  0.015714  0.010000
0.010000  0.015714  0.015714  0.021429  0.010000  0.021429
```

Podemos apreciar que en general la viga tiene una forma homogénea exceptuando algunas secciones donde el algoritmo ha propiciado secciones de radio elevado y poco espesor. Son fácilmente identificables y se podrían modificar si se pretende fabricar la viga.

5.4.2 SOLUCIÓN TEÓRICA

Se realizará asignando un único tamaño de sección:

Al igual que en la herramienta tendremos en cuenta tanto la plasticidad del material como el desplazamiento máximo permitido que se ha establecido (que no supere los 0.003 metros). Calcularemos cuáles serían los parámetros de la sección óptimos en cuanto a masa que cumpla las condiciones:

5.4.2.1 Plastificación del Material

La sección mínima que tendría el material sin que este llegase a plastificar la calcularemos teniendo en cuenta la Ley de Navier. Al ser una sección rectangular hueca la tensión máxima se producirá en los puntos que estén situados más alejados del eje neutro, el eje neutro se encuentra en y=0. Por lo que la tensión normal será tensión principal ya que en los extremos del rectángulo (se supondrá cuadrado para el cálculo de la sección) la tensión tangencial será nula. La tensión de Von Mises será por tanto la tensión normal en dichos puntos.

Sabemos que en una viga con todas las cargas en el plano la tensión normal a la que está sometido el material en un punto será:

$$\sigma_z = \frac{M_x}{Ix} y$$

M_x antihorario en eje x positivo.

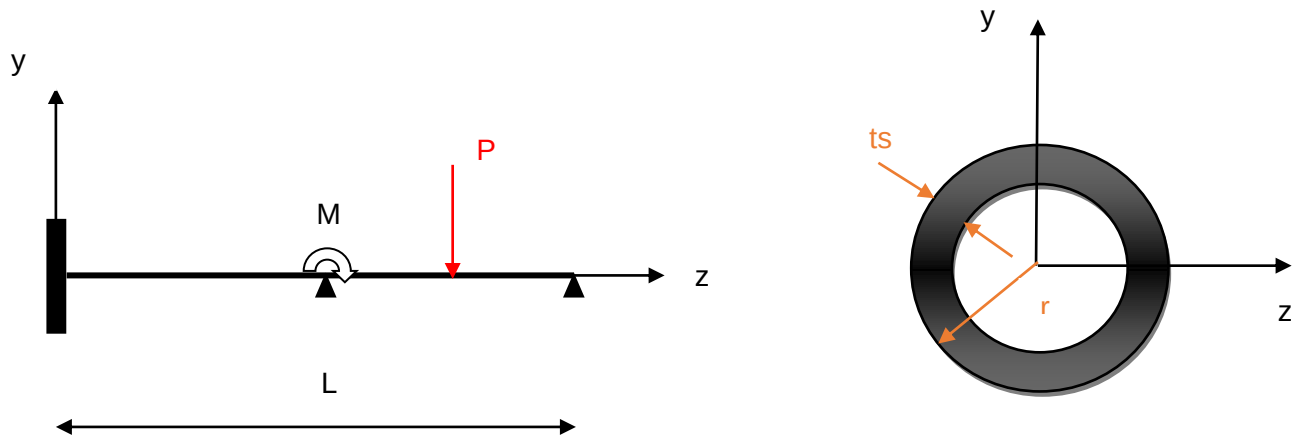


Figura 5.4.2.1.1: Representación Gráfica del Problema

Como se trata de una estructura hiperestática no sabemos a priori donde se encuentran los mayores esfuerzos. Para resolver esta viga se ha empleado el método de liberación de coacciones junto con el teorema de superposición y se han liberado los dos apoyos. Se ha obtenido el siguiente diagrama de momentos flectores donde se puede ver que el momento máximo se produce en el empotramiento con un valor de 42311 N·m. (antihorario).

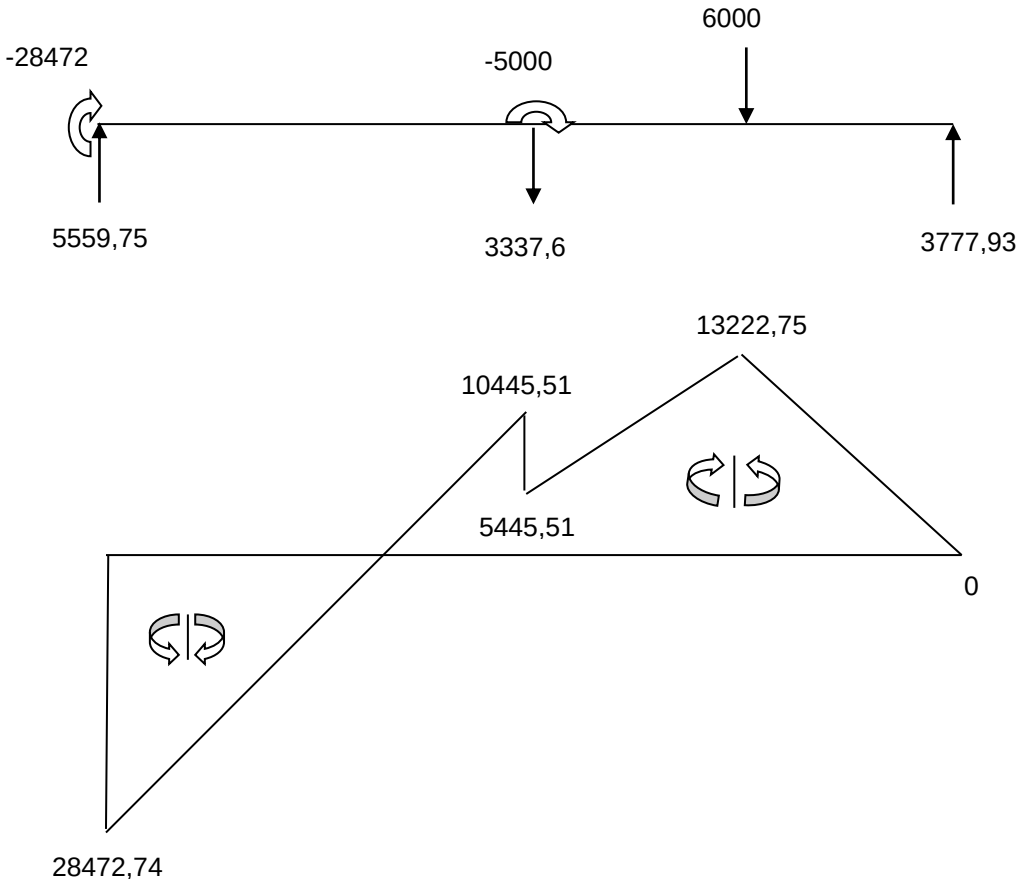


Figura 5.4.2.1.2: Diagrama de Momentos Flectores (Unidades del SI). Punto de corte x=5,12

La tensión máxima a la que puede estar sometido el material es el límite elásticao, σ_y. Los puntos con mayor tensión serán los que estén más alejados del eje neutro (eje neutro tensión normal es igual a 0) por lo tanto teniendo en cuenta que el valor máximo que puede adoptar el radio es "rs" metros, la y_{max} será "rs" metros. El tramo de viga que estará sometido a mayores esfuerzos flectores será el empotramiento donde el momento será 28472,74N·m que corresponde a la reacción del empotramieto. El momento de Inercia de un tubo circular respecto del eje x es $I_x = I_{x1} – I_{x2} = 1/4\pi \cdot rs^4 – 1/4\pi \cdot (rs – ts)^4$

σ_y = 4,50E+08 Pa

y_{max} = rs metros

Si consideramos un espesor tc = rs/10 y despreciando el peso propio de la viga tendremos:

$$\sigma_y = \frac{M_x}{Ix} y \rightarrow rs = \sqrt[4]{\frac{40 \cdot M_x \cdot y}{\sigma_y \cdot \pi}} = \sqrt[3]{\frac{28472{,}74 \cdot 40}{4.5E+08 \cdot \pi}} = 0.093m$$

Por lo tanto la masa de la viga sería:

M= A*L* ρ =567,37 Kg

Habrá que calcular también la masa de la viga teniendo en cuenta el desplazaminto máximo y ver cuál es más restrictiva.

5.4.2.2 Desplazamiento Máximo Permitido

Como tenemos una viga empotrada en un extremo con carga distribuida a lo largo de toda la longitud, deducimos que el desplazamiento máximo se producirá en el extremo donde se encuentra aplicada la carga. Calcularemos el desplazamiento máximo utilizando el Teorema de Mohr:

$$\downarrow V_B = V_{A+}\theta_A \cdot d_{A \rightarrow ejeVB} + \sum_{A}^{B} \left[\frac{(Área\ Mf)_{A \rightarrow Bi}}{EI} d_{CDG \rightarrow ejeVB} \right]$$

En el empotramiento (punto A) los giros y desplazamientos son nulos.

Simplificando la ecuación del Teorema de Mohr y dividiendo en cuatro áreas el área total del momento flector comprendida entre el empotramiento y el punto de aplicación de la carga nos queda que la flecha máxima en el punto donde se encuentra aplicada la carga puntual es:

$$\text{Flecha máxima} = 0{,}003 = \frac{545208{,}09}{2.1e11 \cdot I_x}$$

Si realizamos la misma suposición en cuanto a la relación de radio y espesor que en el apartado de plastificación nos queda:

$$rs = \sqrt[4]{\frac{545208{,}09 \cdot 40}{2{,}1e11 \cdot 0{,}003 \cdot \pi}} = 0{,}324m$$

Por lo tanto la masa de la viga sería:

M= A*L* ρ = 6885,98 Kg

El resultado final que nos da la herramienta es un peso de 1283 kilogramos que comparado con el que obtendríamos con sección constante (6885,98 Kg) es mucho menor. Si además consideramos que en el cálculo analítico no se ha tenido en cuenta el peso propio de la viga los resultados son aún mejores. La parte más negativa del resultado ha sido encontrarnos con dos secciones con un radio muy elevado y espesor pequeño que se salen de la homogeneidad genérica de la pieza y podría dificultar su fabricación, puesto que tenemos los datos de cada una de las secciones no sería difícil localizarlas en un momento dado e intercambiarlas por otras con menor radio y mayor espesor que posean el mismo momento de inercia.

6 CONCLUSIONES

Se ha conseguido alcanzar el objetivo que se pretendía ya que se ha conseguido que la herramienta funcione de forma completamente automatizada y que optimice las estructuras de forma que su peso esté por debajo del peso que tendría en el caso de utilizar una sola sección.

Con lo que se han tenido mayores problemas ha sido a la hora de encontrar una solución que tuviera unas tolerancias buenas entre las distintas secciones para satisfacer un proceso o criterio de fabricabilidad.

Otro problema importante ha sido el alto tiempo computacional que requerían la optimización de cada estructura, ello se debe a que para llegar a una solución óptima se requería que el algoritmo ejecutara numerosas iteraciones debido al alto número de variables. En la solución del CASO 1 por ejemplo se ha necesitado más 50 iteraciones para que el algoritmo llegara a una solución a la que no se le aplicara ninguna restricción y se puede ver que la sección se podría haber optimizado más aún si se aumentara en número de iteraciones

El alto tiempo computacional se debe principalmente la simulación que realiza ABAQUS del problema y en el tiempo de apertura y escritura en archivos externos a MATLAB.

También es importante tener en cuenta una de las limitaciones de los AGs por definición: ”Los AGs sólo se pueden considerar eficientes en comparación con otros métodos estocásticos de búsqueda ciega, pero se puede garantizar que si se encuentra un método heurístico para resolver un problema específico este siempre lo hará mucho más eficientemente que un AG”

Estas son algunas de las limitaciones a las que nos hemos tenido que enfrentar a la hora de utilizar la herramienta creada.

7 TRABAJOS FUTUROS

Teniendo en cuenta las conclusiones mencionadas, puesto que el proyecto consistía en una primera toma de contacto entre la optimización de estructuras sencillas mediante el uso de algoritmos genéticos y elementos finitos me gustaría proponer como trabajos futuros sobre dicha herramienta una ampliación a estructuras más complejas, en las que podamos implementar también rótulas y la creación de pórticos, de manera que nos quede una herramienta mucho más compacta. Y en cuanto resolver el problema del tiempo computacional sería interesante crear nuestro propio programa de elementos finitos utilizando el código de programación de MATLAB, ya que se tratan de estructuras sencillas. En cuanto a la falta de normalización de las secciones calculadas sería interesante añadir restricciones dadas por el código técnico de la edificación.

8 BIBLIOGRAFÍA

ALGORITMOS GENÉTICOS

[1] L. Recalde - *C.P.S. Esquemas algorítmicos - Algoritmos genéticos*

[2] J.T.B. Overvelde. *Learn Abaqus script in one hour*. December 12, 2010

[3] Matías Ison, Jacobo Sitt, Marcos Trevisan. *Guía de la materia Sistemas Complejos*. November 25, 2005. Available: www.df.uba.ar/users/mison/genetico.tar.gz

[4] Marcos Gestal, Daniel Rivero, Juan Ramón Rabuñal, Julián Dorado, Alejandro Pazos. *Introducción a los Algoritmos Genéticos y la Programación Genética*. A Coruña, 2010 Universidade da Coruña

[5] Jesús González García. *Metodología para la asignación de tolerancias y valores nominales a un conjunto de variables dependientes*. Universidad Carlos III de Madrid. Julio 2010. pp 25-54

[6] Carlos Cervigón, Lourdes Araujo. *Programación Evolutiva.* Universidad Complutense de Madrid. 2010. pp.3

ELEMENTOS FINITOS

[7] Santiago Pezzotti, Federico Antico. *Introducción a la Teoría de Elementos Finitos (Tratamiento de la formulación de elementos unidimensionales a partir del método directo).* 2008

[8] A.Carnicero. *Introducción al método de los elementos finitos.*

MATLAB

[9] Javier García de Jalón de la Fuente, Rufino Goñi Lasheras, José María Sarriegui Domínguez, Iñigo Girón Legorburu, Ignacio Rodríguez Garrido, Alfonso Brazález Guerra, Patxi Funes Martínez. *Aprenda MATLAB 4.2 como si estuviera en Primero*. 1 de Septiembre de 1997

[10] Manuel López Martín, José Ángel Acosta Rodríguez. *Manual de Introducción a MATLAB.* 2004

[11] Diego Orlando Barragán Guerrero. *Manual de Interfaz Gráfica de Usuario en MATLAB.*

[12] Lucelly Reyes H. *Interfaz Grafica en MATLAB Guide*

ABAQUS

[13] J.T.B. Overvelde. *Learn ABAQUS script in one hour*. December 12, 2010

[14] Gautam Puri. Python Scripts for ABAQUS: Learn by Example. First Edition 2011,

[15] UC3M. *Manual ABAQUS.* 2012

PYTHON

[16] Andrés Marzal Isabel Gracia. *Introducción a la programación con Python.* Publicacions de la Universitat Jaume I. 2004,

[17] Eugenia Bahit. *Curso: Python para Principiantes*. 2012,

[18] Raúl González Duque. *Python Para Todos*. Creative Commons Reconocimiento 2.5 España, 2001,

[19] Guido van Rossum, *El tutorial de Python*. Septiembre 2009, Available: http://python.org.ar/pyar/Tutorial

[20] Guido van Rossum, Fred L. Drake, Jr., editor. *Guía de aprendizaje de Python*. 16 de octubre de 2000,